VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZŠÍRENIE APACHE TIKA O EXTRAKCIU TEXTU ZO SÚBOROV PRIEMYSLOVÝCH FORMÁTOV

EXTENSION OF APACHE TIKA WITH INDUSTRIAL FILE FORMATS TEXT EXTRACTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **RENÉ REŠETÁR**
AUTHOR

VEDOUCÍ PRÁCE — **RNDr. MAREK RYCHLÝ, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Rešetár René**
Program: Informační technologie
Název: **Rozšíření Apache Tika o extrakci textu ze souborů průmyslových formátů**
Extension of Apache Tika with Industrial File Formats Text Extraction
Kategorie: Informační systémy

Zadání:

1. Prozkoumejte projekt Apache Tika pro extrakci textu z dokumentů, proces extrakce a způsob jeho rozšíření o další formáty dokumentů. Seznamte se s průmyslovými formáty dokumentů s výstupy z laboratorních přístrojů.
2. Po konzultaci s vedoucím navrhněte pro vybrané formáty dokumentů s výstupy z laboratorních přístrojů přidání jejich podpory do Apache Tika včetně způsobu testování úspěšnosti extrakce.
3. Navržené řešení implementujte a důkladně otestujte na vedoucím dodané sadě ukázkových dokumentů.
4. Vyhodnoťte a diskutujte výsledky. Výsledný software publikujte jako open-source.

Literatura:

- MATTMANN, Chris A. a Jukka L. ZITTING. *Tika in action*. Shelter Island, NY: Manning Publications, c2012. ISBN 1935182854. Dostupné z: [https://livebook.manning.com/book/tika-in-action/]
- Get Tika parsing up and running in 5 minutes. *The Apache Software Foundation* [online]. 2020 [cit. 2020-10-26]. Dostupné z: [https://tika.apache.org/1.24.1/parser_guide.html]

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Rychlý Marek, RNDr., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 27. října 2020

Abstrakt

Cieľom bakalárskej práce bolo rozšíriť syntaktické analyzátory projektu Apache Tika o extrakciu tabuliek a dát z priemyslových formátov dokumentov z laboratórnych prístrojov. Tieto dáta majú byť uložené v štruktúrovanom formáte podľa určitej schémy. V teoretickej časti boli preskúmané dodané industriálne formáty, projekt Apache Tika a možnosti jeho rozšírenia. V praktickej časti bol navrhnutý a implementovaný nástroj, ktorý dokumenty pomocou projektu Apache Tika klasifikuje, spracuje, vytvára z nich štruktúrované dáta vo formáte JSON a tie následne validuje. Na záver bola vytvorená sada testov pre overenie a demonštráciu vlastností riešenia.

Abstract

The goal of the bachelor's thesis was to extend the parsers of the Apache Tika project with data and table extraction from industrial document formats from laboratory instruments. These data will be stored in a structured format according to a certain scheme. In the theoretical part, the supplied industrial formats, the Apache Tika project and the possibilities of its expansion were examined. In the practical part, a tool was designed and implemented, which classifies documents using the Apache Tika project, processes them, creates structured data from them in the JSON format and subsequently validates them. Finally, a set of tests was created to verify and demonstrate the properties of the solution.

Kľúčové slová

Java, Apache Tika, Maven, weka, .arff, JSON, pdf, xlsx, csv, software, laboratória, kontrolné laboratória, bez papierové laboratórium, SVP, farmaceutický priemysel, integrita dát, Service Provider, štruktúrované dáta, MIME-typy, extrakcia dát, extrakcia tabuliek

Keywords

Java, Apache Tika, Maven, weka, .arff, JSON, pdf, xlsx, csv, software, laboratories, control laboratories, non-paper laboratories, SVP, farmaceutic industry, data integrity, Service Provider, structured data, MIME-types, data extraction, table extraction

Citácia

REŠETÁR, René. *Rozšírenie Apache Tika o extrakciu textu zo súborov priemyslových formátov*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce RNDr. Marek Rychlý, Ph.D.

Rozšírenie Apache Tika o extrakciu textu zo súborov priemyslových formátov

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána RNDr.MAREK RYCHLÝ,Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
René Rešetár
7. mája 2021

Poďakovanie

Rád by som týmto vyjadril poďakovanie doktorovi Markovi Rychlému za vedenie a cenné rady pri vypracovávaní práce.

Obsah

Kapitola 1

Úvod

Po zásadnej aktualizácií noriem pre akreditované laboratória a to najmä normy ISO 17025, kapitola 7.11, ktorá po novom kladie dôraz na "Riadenie dát a management informácií", bol zvýšený dôraz na splnenie požiadavkov na integritu dát. Tento dôraz je motivovaný najmä faktom, že jedným z najčastejších nedostatkov farmaceutických laboratórií pri inšpekciách Správnej Výrobnej Praxe (SVP), je aktuálne integrita dát. Prakticky to znamená, že norma definuje požiadavky na integritu dát a celkovo sprísňuje riadenie dát a informácií. Validované dátové úložisko (VDÚ) je nástroj, ako tieto požiadavky rýchlo, bezpečne, efektívne a lacno splniť.

Cieľom tejto práce bolo navrhnúť a implementovať nástroj, ktorý má slúžiť na extrakciu dôležitých dát z vybraných formátov dokumentov obsahujúcich výstupy z laboratórnych prístrojov a pridať jeho podporu do Apache Tika. Tento nástroj má využívať a rozširovať funkcionalitu Apache Tika. Jeho výstupom budú dáta v štruktúrovanej podobe, teda dáta zapísané pomocou štrukturovaného jazyka a podľa určitej schémy. Nástroj bude súčasťou spomínaného VDÚ na organizáciu a ukladanie dokumentov a dát v zhode s požiadavkami regulácie a štátnych autorít na farmaceutické (GMP - Good Manufacturing Practice) a ISO 17025 akreditované laboratória.

Nástroj má byť súčasťou VDÚ a preto kapitola 2 popisuje kde presne do tohto projektu spadá. Kapitola 3 potom popisuje MIME-typy, ukazuje dostupné možnosti pre parsovanie a extrakciu dát a popisuje problémy existujúcich riešení. Kapitola 4 nás zoznamuje s Tikou, jej architektúrou a pracovným postupom a popisuje možnosti integrácie. Kapitola 5 prechádza návrh samotného nástroja, požiadavky kladené na jeho funkcionalitu a taktiež opisuje možnosti jeho ďalšieho rozšírenia. Kapitola 6 prechádza k prehľadu použitých technológií, integrácií nástroja a konkrétnej implementácií jeho jednotlivých častí. Kapitola 7 sa zaoberá testovaním a porovnávaním nástroja s Tikou.

Kapitola 2

VDÚ

Nástroj vyvinutý v tejto práci bude použitý v projekte ktorého cieľom je vývoj cloudového a on premise softwarového riešenia "Validovaé dátové úložisko" [22] odteraz VDÚ. VDÚ je určené na organizáciu a ukladanie dokumentov, dát, obrázkov a iného digitálneho obsahu, v zhode s požiadavkami regulácie a štátnych autorít na farmaceutické (GMP) a ISO 17025 akreditované laboratória .

Sú 2 dôvody[1], prečo zákazníci v regulovaných laboratóriách potrebujú produkt typu VDÚ. Potrebujú:

1. ukladať dáta, spracovávať a dolovať dáta zo svojich laboratórií, zo zmluvných laboratórií a potrebujú komunikovať/zdielať dáta v reálnom čase (certifikáty, správy, výsledky analýz, primárne dáta) so svojimi zákazníkmi bez zasielania emailom alebo poštou.

2. splniť požiadavky legislatívy Správnej výrobnej praxe vo farmaceutickom priemysle a v ISO 17025 akreditovaných laboratóriách na integritu dát a "riadenie dát a management informácií". Bez splnenia požiadavkov regulácie laboratórium môže stratiť povolenie ku experimentovaniu/prevádzke, čo ohrozuje jeho existenciu.

Preto bude projekt umožňovať:

- zdieľanie a výmenu informácií zúčastnených strán (vytváranie a nahrávanie dokumentov/súborov, ich ukladanie, klasifikáciu a indexáciu, spracovanie veľkých dát a dolovanie dát)

- splnenie nových legislatívnych požiadaviek na integritu dát (komplexné zabezpečenie, integritu dát podľa princípov ALCOA (Attributable, Legible, Contemporaneous, Original a Accurate), vrátane histórie, revíznych stôp, auditovateľnosti a dohľadateľnosti)

- urýchlenie prípravy na audity a inšpekcie akosti štátnych autorít (automatický reporting a notifikácie, vyhľadávanie a triedenie, delenie štruktúry do logických celkov a správu metadát)

S pomocou nástrojov na analýzu a dolovanie dát môžu užívatelia naviac získať informácie a výstupy, ku ktorým by inak nemali ako inak prísť. Tieto nástroje však najlepšie pracujú so štruktúrovaným textom, ktorý bude výsledný nástroj tejto práce pre konkrétne dokumenty vytvárať.

[1]Informácie z: VUT V BRNĚ. Interní dokumentace projektu Validované datové úložiště. Brno, 2020.

Kapitola 3

Úvod do analýzy dokumentov a štrukturalizácie dát

Táto kapitola predstavuje základné pojmy a teóriu potrebnú pre formalizovaný popis súčasti nástroja. Najprv je potrebné porozumieť digitálnym dokumentom 3.1 a MIME typom 3.2. Následne zadefinovať pojmy ako sú syntaktická analýza dokumentov a štrukturalizácia dát v podkapitole 3.3. Kapitolu zakončuje ukážka niekoľkých existujúcich syntaktických analyzátorov (odteraz parserov) s ich popisom 3.4.

3.1 Úvod do digitálnych dokumentov

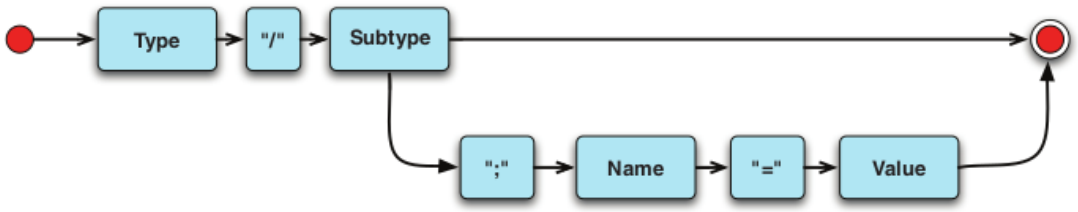
Svet digitálnych dokumentov [13] a ich formátov súborov je ako vesmír, v ktorom každý jeden hovorí iným jazykom. Väčšina programov rozumie iba svojím vlastným súborom alebo malej sade súvisiacich formátov. Zvyčajne sú potrební prekladatelia ako napr. moduly importu alebo zobrazovacie doplnky, keď chce jeden program porozumieť dokumentom vytvoreným inými programami.

Existujú tisíce rôznych súborových formátov používaných v praxi a väčšina z týchto formátov prichádza s variáciami a dialektmi. Napríklad, široko používaný formát PDF sa vyvinul cez osem prírastkových verzií a prešiel rôznymi rozšíreniami za posledných 18 rokov. Našťastie väčšina programov sa nikdy nemusí obávať tohto množenia súborových formátov. Presne tak ako nám stačí hovoriť iba jazykom používaným ľudmi vo vašom okolí. Problém nastáva, keď chcete vytvoriť aplikáciu, ktorá by mala rozumieť väčšine najpoužívanejších súborových formátov. A prvým krokom k vytvoreniu takéhoto nástroja je porozumenie vlastností šírenia súborových formátov, ktoré existujú. K tomu využijeme taxonómiu špecifikovanú v "Multipurposed Internet Mail Extensions" štandarde.

3.2 MIME typy

MIME typy [4][5] alebo po novom **typy internetového média** popisujú najlepší doteraz známy štandard na identifikáciu typov dokumentov. MIME je akronym pre "Multipurpose Internet Mail Extension" čo v slovenčine znamená "viac účelové rozšírenie internetovej pošty" a je to dvojdielny identifikátor formátov súborov. Pôvodne bol vytvorený ako rozšírenie základného formátu emailu a to primárne o používanie širšieho spektra emailových príloh. Teraz je **media typ** základnou stavebnou jednotkou interakcie so súbormi a softvéru každého počítača. Tieto typy hovoria počítaču akú aplikáciu spojiť s akým súborom[13].

Typ média je zložený najmenej z dvoch častí 3.1: **typu, podtypu** (napríklad typ **audio** má podtyp **mp4**, by sa zapísalo ako **audio/mp4**) a jedného či viac nepovinných parametrov. Podtypy typu **text** (textové dáta) majú nepovinný parameter **charset**, ktorý informuje o kódovaní znakov. Podtypy typu **multipart** (dáta zložené z viac častí) často definujú parameter **boundary** popisujúci znakovú sekvenciu ohraničujúcu jednotlivé časti dát.



Obr. 3.1: Diagram syntaxe mien typov média. Prevzaté z [13]

IANA "Internet Assigned Numbers Authority" [25] je organizácia pre štandardy, ktorá okrem iného dohliada na globálne prideľovanie IP adries a prideľovanie autonómnych čísel systému. Taktiež spravuje register typov médií a kódovania znakov a zverejňuje ich zoznam. Tu sú niektoré významné typy a podtypy [4][5]:

- typ **application**, viacúčelové (aplikačné) súbory:
  - **application/pdf** Portable Document Format (PDF) [27], používa sa na ukladanie dokumentov nezávisle od softvéru, hardvéru a operačného systému na ktorom boli vytvorené a taktiež na zariadení, na ktorom sú zobrazované. Súbory typu PDF môžu obsahovať text a obrázky taktiež aj interaktívne formuláre, videá, animácie, 3D grafiku, zvukové stopy a elektronické podpisy, pričom primárnym účelom formátu je zabezpečiť, že sa dokument na všetkých zariadeniach zobrazí rovnako;
  - **application/zip** súbory v archíve ZIP. ZIP [28] je jeden z najbežnejšie používaných formátov komprimovaných súborov. Je univerzálne používaný na agregáciu, kompresiu a šifrovanie súborov do jedného kontajner.
  - **image/png** Portable Network Graphics
- typ **image**, obrázkové formáty:
  - **image/gif** obrázok GIF
  - **image/png** Portable Network Graphics

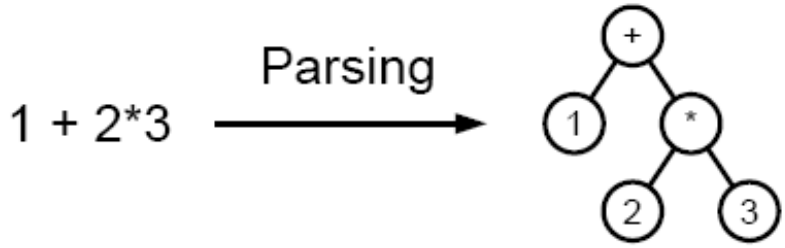
3.3 Syntaktická analýza a štrukturalizácia objemných dát

Syntaktická analýza [10] alebo parsovanie je v počítačovej vede a lingvistike proces analýzy sekvencie tokenov na určenie ich gramatickej štruktúry s ohľadom na danú formálnu gramatiku. Pri syntaktickej analýze sa vstupný text spravidla transformuje na určité dátové štruktúry, väčšinou syntaktický strom, ako je možné vidieť na obrázku 3.2. Syntaktická analýza je dôležitou časťou spracovania programov pri ich preklade (kompilácií) alebo interpretácií.

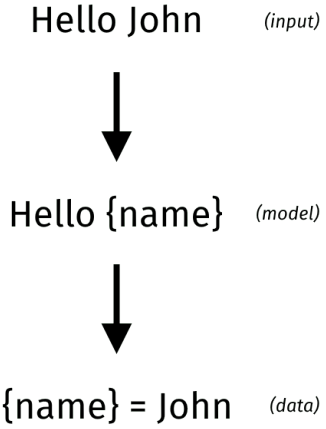
Tento prístup sa používa aj pri spracovaní ľudského jazyka za pomoci parserov, ktoré dokážu ľudský jazyk syntakticky analyzovať. Táto úloha je veľmi sťažená skutočnosťou, že stavba ľudského jazyka je spravidla veľmi nejednoznačná. Pre syntaktickú analýzu ľudského jazyka musí byť najprv stanovená príslušná formálna gramatika s podobným princípom ako ukazuje obrázok 3.3. Voľba jej syntaxe závisí na zámeroch ako lingvistických, tak aj implementačných.

Zjednodušene povedané, je parsovanie [23] analýza vstupu pre zorganizovanie dát podľa určených pravidiel. Tieto pravidlá určujeme aj napríklad pomocou regulárnych výrazov, kde môžeme použiť Kleeneho hviezdičku (*) ako pravidlo indikujúce, že nejaký element môže byť prítomný nula alebo nekonečne veľa krát.

Syntaktický analyzátor [10] alebo parser je program, ktorý vykonáva syntaktickú analýzu, či už počítačového alebo ľudského jazyka.



Obr. 3.2: Jednoduchá ukážka prevodu aritmetického výrazu **1 + 2 * 3** na syntaktický strom. Prevzaté z [10]



Obr. 3.3: Jednoduchá ukážka princípu parsovania. Gramatika určuje, že po "Hello" kľúčovom slove nasleduje hodnota *name*, ktorá je dôležitá. Parser teda zaznamená hodnotu "John" ako *name*. Prevzaté z [23].

Štrukturalizácia dát

Aby sa údaje alebo informácie [9] (získané napr. aj syntaktickou analýzou), preukázali ako užitočné a mohli byť ďalej použité (napr. v hĺbkovej analýze dát), musíme vyvinúť štandardy a organizačné štruktúry a za ich použitia tieto údaje usporiadať do ucelenej štruktúry.

Za týmto účelom boli v priebehu desaťročí vyvinuté **relačné databázy**[1]. Avšak relačné databázy [1] majú „impedančný nesúlad“ medzi programovacími jazykmi a dátovými štruktúrami v databáze. To znamená, že na mapovanie alebo využitie relačných údajov v rôznych projektoch je potrebné použiť ďalšie programovacie metódy, napr. Objektovo orientované programy. Tu môžu pomôcť polo štrukturované jazyky ako JSON alebo XML.

JSON

JavaScript Object Notation (JSON) [6] je textový formát pre serializáciu štruktúrovaných údajov. Je to polo štruktúrovaný jazyk[2] odvodený z objektových literálov JavaScriptu navrhnutý tak, aby bol čitateľný pre ľudí a štruktúrovaný do vnorených množín (objekty, polia). JSON môže predstavovať štyri primitívne typy:

- *reťazce* (“Táto veta je reťazec”),
- *čísla* (56, -698, 2.5),
- *pravdivostné hodnoty* (true, false)
- a *null*

a dva štruktúrované typy:

- *objekty* {“property”: “value”}
- a *polia* [1, 2, 3, 589]

JSON Schema

JSON Schema [26] je slovník (slovná zásoba), ktorý umožňuje anotovať a overovať JSON dokumenty. Výhody jeho použitia:

- opisuje existujúce formáty údajov
- poskytuje jasnú dokumentáciu čitateľnú pre človeka i stroj
- validuje dáta, čo je užitočné pre automatické testovania a zaručenie kvality údajov predložených klientom

3.4 Existujúce nástroje

Táto podkapitola popisuje existujúce nástroje na parsovanie dokumentov a štrukturalizáciu dát. Sústredí sa najmä na nástroje spracúvajúce PDF dokumenty, keďže ich parsovanie predstavuje asi najväčšiu výzvu.

[1] Relačná databáza sa vyznačuje údajmi, ktoré sú uložené v tabuľkách. Je založená z viacerých tabuliek, ktoré sú navzájom prepojené tzv. musia mať medzi sebou vzťahy, väzby

[2] Forma jazyku štruktúrovaných dát, ktorá sa neriadi tabuľkovou štruktúrou dátových modelov spojených s relačnými databázami alebo inými formami údajových tabuliek, ale napriek tomu obsahuje tagy alebo iné značky na rozoznanie sémantických prvkov a vynucovanie hierarchií záznamov a polí v dátach

Tabula

Tabula [2] je nástroj na oslobodenie tabuľkových údajov uväznených v súboroch PDF. Bola vytvorená novinármi pre novinárov a hocikoho iného pracujúceho s dátami uzamknutými v PDF súboroch. Dokáže spracovať iba textovo založené PDF súbory, nie naskenované dokumenty. Poskytuje grafické rozhranie v prehliadači alebo možnosť použitia v príkazovom riadku pomocou knižnici *tabula-java*[3], na ktorej je postavená. V grafickom rozhraní je možné ručne označiť časť, na ktorú sa má Tabula zamerať. V rozhraní príkazového riadku je potrebné zadať súradnice (napr. v takomto formáte -a =269.875,12.75,790.5,56).

Hlavné výhody tohto nástroja sú:

- Dostupnosť (nástroj je opensource)
- Ak je na vstupe textovo založený PDF súbor, Tabula vytvára veľmi kvalitné výstupy, hlavne v CSV a TSV[4] formáte.
- Pri návrhu sa myslelo na bezpečnosť dát. Všetko spracovanie dát prebieha na lokálnom počítači aj pri použití rozhrania v prehliadači.

Medzi najväčšie nevýhody by som zaradil:

- Nástroj dokáže spracovať iba PDF formát
- Aj keď sú vytvorené dáta vo formáte CSV a TSV pekné, nedá sa to isté povedať o JSON výstupe, ktorý obsahuje aj súradnice jednotlivých častí tabuľky v dokumente a tým znemožňuje ich použitie.
- Ak je nástroj použitý v prehliadači tak je výber oblasti na extrakciu bezproblémový. Avšak zadávať vždy túto oblasť v príkazovom môže byť problematické. Ani prepínač *-g* pre náhodnú predpoveď tejto oblasti to nezlepšuje, keďže nie je dostatočne spoľahlivý a často pridá aj nepodstatné časti dokumentu.

pdf2json

Je to [14] modul node.js[5], ktorý analyzuje a prevádza PDF z binárneho formátu do formátu JSON. Je vytvorený pomocou súboru pdf.js a rozširuje ho o interaktívne prvky formulára a analýzu textového obsahu mimo prehliadača.

Cieľom je povoliť syntaktickú analýzu súborov PDF na strane servera s interaktívnymi prvkami formulárov, keď sú zabalené vo webovej službe, a tiež povoliť syntaktickú analýzu lokálneho súboru PDF na súbor JSON, keď sa používa ako obslužný program príkazového riadku. Jeho výstupy však pri mojich pokusoch o extrahovanie boli neúplné.

Docsumo

Softvér Document AI [18] s technológiou Intelligent OCR pomáha previesť neštruktúrované dokumenty, ako sú napríklad platby, faktúry a bankové výpisy, na použiteľné údaje. Pracuje s dokumentmi v akomkoľvek formáte s minimálnym nastavením.

[3] `https://github.com/tabulapdf/tabula-java`

[4] *Comma separated values* a *tab separated values*, sú neštrukturované formáty dát. Ako názvy naznačujú, dáta sú oddelené čiarkou alebo tabulátorom.

[5] `https://nodejs.org/en/`

Docsumo bol zavedený ako jeden z najefektívnejších a najlepšie prispôsobených nástrojov na snímanie údajov o faktúrach. Okrem možnosti použitia online vo webovom prehliadači, ponúka aj API.

Tento nástroj je veľmi presný (podľa stránky ponúka presnosť viac ako 98%) a rýchly, avšak je spoplatnený.

3.4.1 Zhrnutie

Každý z týchto parserov produkuje podobné a pekné výstupy. Problém však nastáva ak je potrebné parsovať dokumenty a súbory rôznych typov. To by znamenalo volať pre každý MIME typ iný parser. To je možné robiť buď ručne alebo automatizovane pomocou detektoru MIME typov, ktorý by rozoznal o aký typ sa jedná a následne dokument/súbor delegoval príslušnému parseru. Aj po úspešnom preparsovaní však výstupy týchto spomenutých nástrojov nie sú dokonalé (okrem *Docsumo*, ktorý však nie je voľne dostupný). To je spôsobené hlavne ich všeobecným prístupom ku všetkým dokumentom. Snažia sa zo všetkých dokumentov získať čo najlepšie dáta avšak k tomu je ešte stále potrebný jedinečný prístup, zodpovedajúci stavbe dokumentu a jeho obsahu.

Kapitola 4

Apache Tika ako riešenie

Táto kapitola popisuje nástroj Apache Tika[1] a dôvody, prečo je vhodný ako prvý krok od ktorého sa odraziť v tejto práci. Prvá podkapitola predstavuje Tiku a jej dve hlavné rozhrania 4.1. Nasleduje jej interakcia s MIME typmi 4.3. Ďalšia podkapitola ukazuje jej architektúru na aplikačnej úrovni a jej pracovný postup 4.1. Predposledná podkapitola sa zaoberá výstupmi Tiky pre dodané súbory 4.5. Nakoniec sú popísané možnosti rozšírenia Tiky 4.6. Väčšina tejto kapitoly je citovaná z [13].

4.1 O Apache Tika

V predošlých podkapitolách 3.2 bola zdôraznená dôležitosť rozoznávania jednotlivých typov súborov a Apache Tika [13] poskytuje funkcie na identifikáciu viac ako 1400 typov súborov z taxonómie MIME typov internetového úradu pre pridelené čísla. Pre väčšinu najbežnejších a populárnejších formátov Tika potom poskytuje možnosti extrakcie obsahu, extrakcie metadát a identifikácie jazyka. Môže tiež získavať text z obrázkov pomocou OCR softvéru Tesseract[2].

Tento nástroj je rámec na detekciu a analýzu obsahu, napísaný v prostredí Java, riadený nadáciou Apache Software Foundation[3]. Okrem poskytovania knižnice Java má aj vydania servera a príkazového riadku vhodné na použitie z iných programovacích jazykov. Tika tieto možnosti ponúka v rámci dvoch hlavných rozhraní, *Parser a Detector* 4.4.

4.2 Tika architektúra a workflow

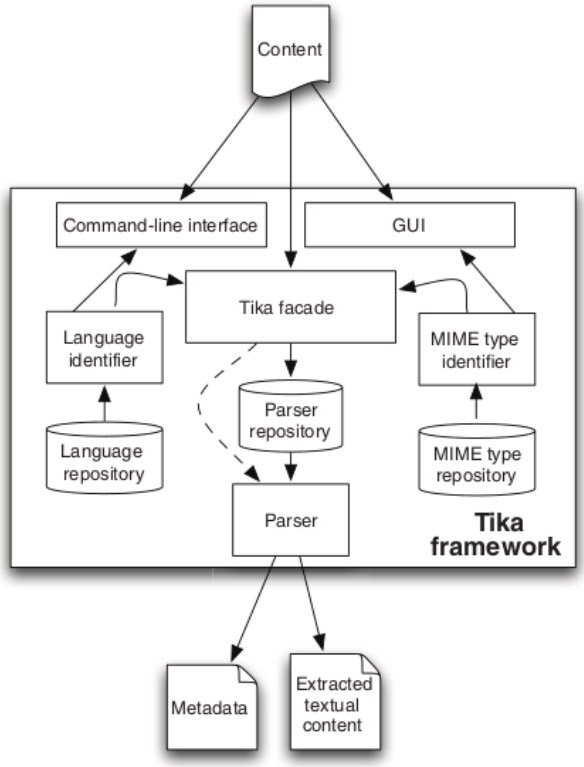
High-level architektúra

Architektúra Tiky [13] na aplikačnej úrovni 4.1 sa dá zhrnúť v týchto kľúčových prvkoch. Parser Rámec (stredná-dolná časť diagramu), MIME detekčný mechanizmus (pravá strana diagramu), detekcia jazykov (ľavá strana diagramu), a fasáda (stred diagramu), ktorá spája všetky komponenty dokopy. Externé rozhrania, čiže príkazový riadok (hore-vľavo na diagrame) a grafické užívateľské rozhranie (hore-vpravo na diagrame), umožňujú užívateľom integrovať Tiku do svojich skriptov a aplikácií a komunikovať s ňou.

[1] `https://tika.apache.org/`

[2] Tesseract [20] je optické rozpoznávanie znakov pre rôzne operačné systémy. Je to slobodný softvér vydaný na základe licencie Apache dostupný na `https://github.com/tesseract-ocr/tesseract`

[3] `https://www.apache.org/`



Obr. 4.1: Vysoko-úrovňová architektúra Apache Tika. Prevzaté z [13]

Workflow

Tika ponúka tri hlavné užívateľské rozhrania pre jej používanie. Pomocou grafického rozhrania alebo v príkazovom riadku lokálne či cez server. Zatiaľ čo grafické rozhranie je pre bežného užívateľa prístupnejšie, nedovoľuje užívateľovi všetko čo príkazový riadok. V tejto práci je za požitie Tiky vždy brané jej lokálne volanie cez príkazový riadok.

Na obrázku 4.2 môžeme vidieť základné spracovanie dokumentu Tikou za účelom získania textového obsahu.

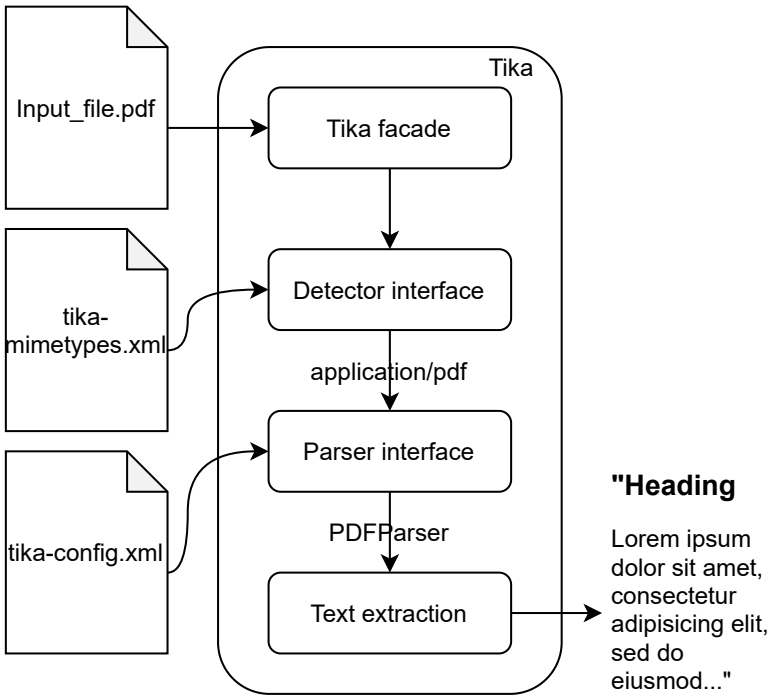
1. Pre získanie základného textového výstupu pri použití Tiky cez príkazový riadok je nutné zadať argument *-t /–text*. Tika spracuje tento argument a nastaví spracovanie výstupu parsera na textové. Tika ďalej použije implicitnú konfiguráciu obsiahnutú v jej zdrojových súboroch *tika-config.xml*, *tika-mimetypes.xml*. O to všetko sa stará **Tika facade**.

2. Následne sa rozhranie detektor (**Detector interface**) pustí do rozpoznania MIME typu dokumentu. Ak má typ konkrétneho dokumentu záznam v *tika-mimetypes.xml* súbore, detektor ho rozpozná.

3. Zistený typ sa odovzdáva Parser rozhraniu (**Parser interface**), ktorý pre neho následne vyberie najlepší možný parser pomocou konfigurácie v *tika-config.xml* súbore a tento parser načíta pomocou *Parser* rozhrania poskytovateľa služieb

4. Vybraný parser **extrahuje text** z dokumentu a keďže bol formát výstupu nastavený na *text*, vypíše ho na štandardný výstup v normálnom textovom formáte.

.



Obr. 4.2: Jednoduchý diagram zobrazujúci Tika pracovný postup pre základnú extrakciu textu.

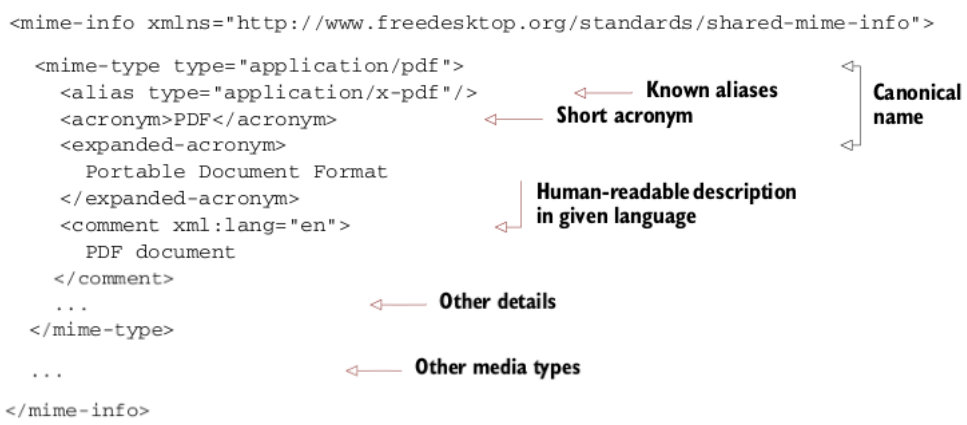
4.3 MIME typy v Apache Tika

Tika udržuje bohatú, ľahko rozšíriteľnú a zrozumiteľnú MIME-info databázu [13] internú pre projekt, čím znižuje externé závislosti na existujúcich registroch. Tika projekt si okrem oboch oficiálnych IANA-registrovaných typov a iných známych typov, ktoré sú v praxi používané udržuje vlastný register typu média. Tento register taktiež sleduje súvisiace informácie ako sú vzťahy typu a kľúčové charakteristiky súborového formátu identifikovaného pomocou média typu. Bohatosť metódy na rozpoznávanie typov je priamo závislá na tejto Tika MIME-info databáze, ktorú si v tejto podkapitole priblížime.

Unixové systémy nemali v minulosti žiaden štandard pre zdieľanie informácií o typoch dokumentov medzi aplikáciami. Preto prišli s riešením nezávislým na platforme. To je Zdieľaná MIME-info Databáza (Shared MIME-info Database), ktorá mimo iné veci definuje XML formát informácií o média typoch. Tento formát, ukázaný na 4.3, je taktiež používaný Tikou. Túto databázu môžeme nájsť v *tika-mimetypes.xml* v zdrojových súboroch projektu Tika.

Tento súbor sa skladá z *mime-type* záznamov, ktorý každý popisuje jeden typ média. Záznam typu špecifikuje oficiálne meno typu ako aj všetky jeho známe druhé mená (tzv. aliasy). Napríklad, ako vidíme na 4.3, niektoré oficiálne typy sú známe pod experimentálnymi *x-** menami, ktoré predchádzali ich oficiálnej registrácií. Záznam môže taktiež obsahovať neformálne mená typu, ktoré sú používané v ľudskej komunikácií[4].

[4]V bežnej komunikácií namiesto *"application/pdf"* normálne povieme, *"PDF dokument"*



Obr. 4.3: Ukážka Zdieľanej MIME-info databázy prevzatá z [13]

Od uchovávania typov prejdime rovno k ich rozoznávaniu. To je vo všeobecnosti založené na použití charakteristických vlastnostiach digitálnych dokumentov:

Názov dokumentu

Začína s **globálnymi vzormi súborov**, jednou s najrozšírenejších a najľahších metód pre detekciu typov média. Je to jednoduché, stačí sa pozrieť na názov súboru. Najmodernejšie operačné systémy a aplikácie používajú súborové rozšírenia ako **.txt** alebo **.png** aby indikovali typ súboru.

Nápoveď zanechaná tvorcom dokumentu

V prípadoch, že názov súboru nie je dostupný sa používajú **content type hints**, teda akési ukazovatele alebo rady pre typ obsahu. To je bežné keď sa súbor nachádza v databáze, ku ktorej pristupujeme cez sieť, alebo je pridaný ako súčasť iného súboru. V týchto prípadoch je pre dokument typické byť asociovaným s nejakou externou informáciou o type, najčastejšie explicitným média typom. Napríklad, HTTP protokol používaný webovými prehliadačmi na požiadavky HTML stránok a iných dokumentov z webových serverov špecifikuje *Content-Type* v hlavičke.

Magický byte

Aj keď sú predošlé metódy dosť presné, nedá sa na nich vždy spoľahnúť. Niekedy nie sú externé informácie dostupné alebo sú nesprávne, takže jediný spôsob ako správne určiť typ súboru, je pozrieť sa doň a skúsiť detegovať podľa jeho obsahu. Skoro všetky súborové formáty majú nejakú charakteristickú vlastnosť alebo vzor, ktorý môže byť detegovaný keď sa pozrieme na bytový obsah súboru. Veľa formátov dokonca obsahuje **magický byte**, teda prefix ktorý je určený k správnej identifikácií súborového formátu. GIF obrázky napríklad začínajú ASCII charaktermi **GIF87a** alebo **GIF89a** závisiac podľa verzie použitého GIF formátu. Pre PDF dokument je to napríklad **%PDF-** v ASCII alebo pre JPEG obrázok je to **FF D8 FF** v hex kódovaní.

Kódovanie znakov

Veľký problém textu je, že existuje veľa spôsobov jeho reprezentácie ako bytov. Tieto reprezentácie sa volajú *character encodings* (kódovanie znakov) a sú ich používané stovky. *text/plain* média typ je často sprevádzaný *charset* parametrom, ktorý indikuje kódovanie znakov v textovom dokumente. Ale aj keď je táto informácia dostupná, často je nesprávna. **BOM markery** sú najľahším spôsobom detegovania kódovania znakov. *Unicode* kódovanie používa *byte order mark* BOM aby indikovali poradie v akom sú kódované byty zoradené. **U+FFF** *Unicode* charakter je rezervovaný pre tento účel. Ďalšie spôsoby sú pomocou **frekvencie bytov** alebo **štatistického porovnávania**.

Iné mechanizmy

Niektoré formáty dokumentov sú založené na viac generických formátoch ako **ZIP** archívy, **XML**, alebo **Microsoft's format for Object Linking (OLE)**. Aj keď sa takýto formát kontajnera dá ľahko rozpoznať pomocou magických bytov, môže byť ťažké určiť, či sa formát používa pre hosťovanie konkrétnejšieho druhu dokumentu. Preto je potrebné formát preparsovať a zistiť, či sa obsah zhoduje s nejakým špecifickejším typom.

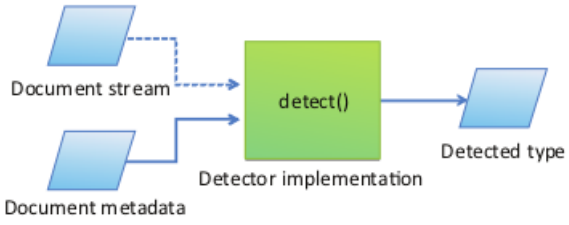
Kombinované heuristiky

Tika detektory a iné vlastné detektory sú konštante vyvíjané vždy keď Tika narazí na nový formát dokumentu, ktorý sa nedá detegovať jedným alebo viacerými jednoduchšími mechanizmami spomínanými vyššie. Jednotne tieto mechanizmy neponúkajú veľkú percentuálnu úspešnosť správneho rozpoznania. Avšak kombinovaním rôznych dostupných heuristík môžeme dosiahnuť veľkú presnosť určovania skoro všetkých typov dokumentov. Tika to našťastie všetko robí za nás.

4.4 Rozhrania Parser a Detector

Detector rozhranie

Je základ pre väčšinu obsahového rozpoznávania [13] v Apache Tike a to platí aj pre rozoznávanie MIME typov. Rozhranie detektora určuje všeobecné API pre algoritmy detekcie typov. Metóda *detect* definovaná v tomto rozhraní deteguje typ dokumentu na základe surového toku bajtov dokumentu a akýchkoľvek dostupných metadát dokumentu ako je vidno na obrázku 4.4.



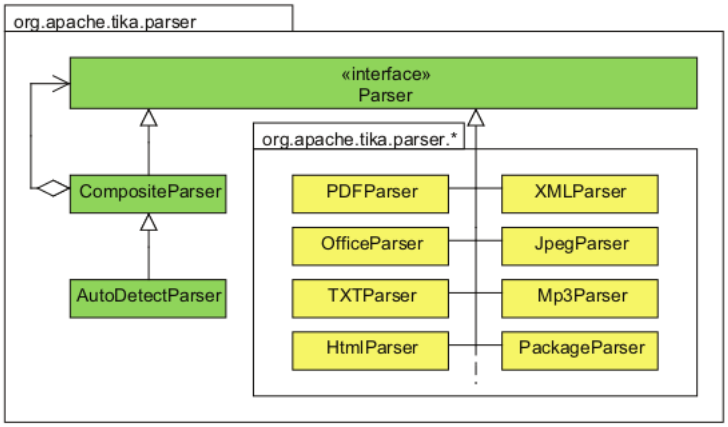
Obr. 4.4: Ilustračná ukážka *detect* metódy. Prevzaté z [13]

Rôzne implementácie detektorov majú rôzne prístupy. Zatiaľ čo jeden sa môže pozerať na tok bajtov pre známy bajtový vzor, iný môže využívať iný s prístupov, ktoré sú popísané v 4.3. Detektor však vždy musí byť pripravený na absenciu hociktorého z týchto vstupov.

Parser rozhranie

Rozhranie *org.apache.tika.parser.Parser* [13] je základným kameňom rozhrania Tika API. Na obrázku 4.5 je zobrazená ukážka jeho architektúry. Používa sa ako verejné rozhranie, ku ktorému majú prístup klientske aplikácie, aj ako rozširujúci bod implementovaný modulmi plug-in analyzátora, ktoré každé podporuje iný formát dokumentu. Toto všetko pomocou jednej jednoduchej metódy *parse*.

Rozhranie bolo v priebehu rokov starostlivo vytvorené na základe skúseností a spätnej väzby od mnohých rôznych používateľov a prípadov použitia.



Obr. 4.5: Ukážka *org.apache.tika.parser.Parser* rozhrania prevzatá z [13]

4.5 Tika a výstupy pre dodané dokumenty

Aj napriek možnosti Apache Tika vytvoriť na výstupe dáta vo formáte XHTML alebo HTML, čiže ako štruktúrované, nie sú tieto dáta vo väčšine prípadov dostatočne presné. To je preto že pre všetky súbory rovnakého typu sa používa vždy ten istý parser (pre všetky PDF súbory PDFParser, pre všetky CSV súbory TextAndCSVParser) a tieto parsere pochopiteľne nedokážu rozlišovať dôležité časti dokumentov (tabuľky, atribúty a ich hodnoty,...). Preto je potrebné na túto úlohu rozšíriť funkčnosť Tiky.

Takéto rozšírenie by muselo okrem typov dokumentov rozlišovať aj jednotlivé dokumenty podľa obsahu a zavolať pre každý druh vlastný parser.

4.6 Možnosti rozšírenia Tiky

Apache Tika [13] je možné rozšíriť o:

1. nové MIME-typy
2. detekciu vlastných typov
3. vlastný parser

Keďže sa tento nástroj a práca zaoberajú typmi, ktoré sú všeobecne a teda aj Tike známe, prvé dve možnosti nie sú pre túto prácu dôležité.

Písanie Vlastného parsera

Ako je spomenuté vyššie 4.4, Tika používa pre parsovanie dokumentov *Parser* rozhranie. Prvým krokom k umožneniu extrakcie informácií z nových druhov dokumentov je implementovanie novej *Parser* triedy. Druhá možnosť je rozšírenie existujúceho parsera.

Pre účely tejto práce bolo výhodnejšie držať sa prvého prístupu. Nový Tika Parser musí byť rozšírením triedy AbstractParser. Okrem toho musí implementovať metódy *parse()*, ktorá oobsahuje všetku funkcionalitu parsera a *getSupportedTypes()*, ktorá slúži pre spojenie parserov s MIME-typmi a vracia MIME typy, pre ktoré by mohol byť konkrétny parser načítaný ak sa pre daný dokument nenájde lepší parser.

Service Provider Interface

Všetky parsere v Tike [13] sú distribuované pomocou rozhrania poskytovateľa služieb, *angl. Service Provider Interface (SPI)* [21]. Za službu (Service) sa v Jave považuje definovaná množina rozhraní alebo tried. V tomto prípade je to rozhranie *org.apache.tika.parser.Parser* a triedy v Tike, ktoré s ním súvisia. Poskytovateľ služby (Service Provider) je špecifická implementácia služby. V Tike ich napríklad predstavujú všetky triedy na obrázku 4.5 v balíku *org.apache.tika.parser.**, V implementácií tieto triedy konkrétne rozširujú *AbstractParser* a až ten rozširuje *Parser* rozhranie.

Poskytovatelia konkrétnej služby sú nakonfigurovaní v súbore *konfigurácie poskytovateľov* umiestnenom v META-INF/services adresári. Súbor konfigurácie poskytovateľa je plne kvalifikované meno SPI a jeho obsahom sú plne kvalifikované mená SPI implementácií. Pre *Parser* SPI sa tento súbor volá *org.apache.tika.parser.Parser* a uvádza plne kvalifikované mená tried, ktoré všetky rozširujú konkrétne SPI. Tieto triedy sú inštancované použitím predvoleného konštruktéra keď je poskytovateľ tejto služby vyhľadaný.

Zapojenie nového parseru do Tiky

Tika používa mechanizmus poskytovateľa služieb (Service Provider mechanism) aby načítala všetky dostupné implementácie z *classpath*. Aby sa Tike povedalo o existencii nového parseru, je nutné umiestniť jeho skompilovanú triedu do JAR súboru spolu s META-INF/services/org.apache.tika.parser.Parser súborom, ktorý uvádza celé kvalifikované meno triedy tohto nového parseru na individuálnom riadku. Keď tento JAR archív pridáme do classpath počítača, Tika automaticky začne používať náš parser.

4.7 TikaConfig

Problém nastáva keď existujú dva parsere, ktoré obe tvrdia, že parsujú ten istý typ dokumentu. V tomto prípade Tika [13] ponúka užívateľovi vysoký level kontroly nad tým, ktorý parser bude a ktorý nebude používaný, v akom poradí preferencií atď. Taktiež je možné prepísať iba určité časti, ako napríklad, zmeniť iba parsovanie pre PDF súbory a zbytok ponechať nezmenený. Tieto zmeny sa nesťahujú iba na Parser rozhranie, ale dajú sa aplikovať aj pre Detector rozhranie, Mime typy, LanguageDetector rozhranie, prekladače či ServiceProvider rozhranie.

K týmto úpravám sa využíva Tika Config XML súbor. Pre prepísanie určitého predvoleného chovania, je potrebné do konfigurácie pridať *DefaultParser* s vylúčenými typmi, pre ktoré chcete toto chovanie zmeniť. Následne je potrebné pridať definíciu nového parsera. Na úplne zabránenie používania *DefaultParsera*, ho stačí jednoducho vynechať z konfigurácie

a vymenovať všetky ostatné parseri, ktoré sa budú namiesto neho používať ako je možné vidieť na výpise 1.

Podobné rozšírenia sa dajú zaviesť aj do Tika MIME-info databázy 4.3, ktorú je možné rozšíriť o nové alebo vlastné vytvorené typy súborov. To sa dá jednoducho ich vpísaním do *tika-mimetypes.xml* súboru, ale bezpečnejší a odporúčanejší prístup je vytvorenie *custom-mimetypes.xml* súboru, ktorý sa potom Tike predá pomocou argumentu *–config=<custom-mimetypes.xml>*. Na vstupe tohto argumentu sa očakáva súbor formátovaný rovnakým spôsobom ako je formátovaný *tika-config.xml* implicitný konfiguračný súbor.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<properties>
  <!-- https://tika.apache.org/1.24.1/configuring.html -->
  <!-- to utilize also dynamically loaded services and log a warning when
  ↪  providers fail to initialize -->
  <service-loader dynamic="true" loadErrorHandler="WARN"/>
  <parsers>
    <parser class="org.apache.tika.parser.DefaultParser">
      <mime-exclude>application/pdf</mime-exclude>
      <mime-exclude>
      application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
      </mime-exclude>
      <mime-exclude>text/csv</mime-exclude>
      <parser-exclude class="org.apache.tika.parser.pdf.PDFParser"/>
      <parser-exclude
       ↪  class="org.apache.tika.parser.microsoft.ooxml.OOXMLParser"/>
      <parser-exclude class="org.apache.tika.parser.csv.TextAndCSVParser"/>
    </parser>
    <parser class="re.deciders.PDFDecider">
      <mime>application/pdf</mime>
    </parser>
    <parser class="re.deciders.XLSXDecider">
      <mime>
      application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
      </mime>
    </parser>
    <parser class="re.deciders.CSVDecider">
      <mime>text/csv</mime>
    </parser>
  </parsers>
</properties>
```

Výpis 1: Obsah súboru *tika-config-bc.xml*, využívanom v tomto projekte. Súbor zabezpečuje aby sa namiesto *PDFParser* volal *PDFDecider* pre *pdf* súbory, namiesto *OOXMLParser* volal *XLSXDecider* pre *vnd.openxmlformats-officedocument.spreadsheetml.sheet* súbory a namiesto *TextAndCSVParser* volal *CDVDecider* pre *csv* súbory.

Kapitola 5

Návrh a architektúra nástroja ReFormator

ReFormator je názov výsledného nástroja a v tejto kapitole je prezentovaný jeho celkový návrh a architektúra. Prvá podkapitola uvádza požiadavky kladené na nástroj 5.1. V druhej podkapitole sú z týchto požiadavok vyvodené dôležité poznatky pre návrh 5.2. Následujúca podkapitola sa zaoberá architektúrou nástroja 5.3. Ďalej je opísané, kam presne do pracovného postupu Apache Tika sa vyvinutý nástroj zapojí a tiež ako bude prebiehať spracovanie dokumentu v samotnom nástroji 5.4.

5.1 Požiadavky na funkcionalitu nástroja

Ako bolo spomenuté v Úvode 1, nástroj by mal rozširovať funkcionalitu Apache Tika a vo všeobecnosti slúžiť na extrakciu a následnú štrukturalizáciu dát. V tejto podkapitole si priblížime všetky špecifické požiadavky, ktoré boli brané do úvahy pri návrhu.

- **Rozšírenie funkcionality Apache Tika.** ReFormator by mal byť navrhnutý ako pridanie podpory dodaných dokumentov do Apache Tika.
- **Extrakcia** dôležitých dát z dodaných dokumentov, kde za dôležité dáta v tomto prípade považujeme: tabuľky, dáta typu *key=value* a metadáta o súbore.
- Dodané dokumenty sú tvorené rôznymi prístrojmi, ktoré využívajú pri ich zostavovaní rôzne nástroje a prístupy. To znamená, že jedna dodaná skupina dokumentov sa líši od druhej a vyžaduje osobitý prístup pre spracovanie, teda vlastný parser. Z toho vyplýva potreba **klasifikácie** vstupného dokumentu a jeho následné posunutie správnemu parseru.
- **Štrukturalizácia** dát na výstupe. Teda vytvorenie súboru obsahujúceho extrahované dáta korektne prepísané v štruktúrovanom jazyku.
- **Kontrola validity** pre každý výstup. Nutnosť skladanie výsledných dát podľa schémy a následne ich voči nej skontrolovať.
- **Rozšíriteľnosť** o podporu nových typov dokumentov. Teda pridanie nových parserov, nových pravidiel pre rozpoznanie typu dokumentu (prípadne nahradiť súbor s existujúcimi pravidlami vlastným súborom).

- Nástroj by mal byť patrične **otestovaný** na dodaných dokumentoch aby sa zaručila jeho funkčnosť.

5.2 Funkcionalita nástroja

Z požiadavok uvedených v predchádzajúcej kapitole 5.1 boli vyvedené poznatky dôležité pre návrh architektúry nástroja.

Napojenie na Tika workflow

Predovšetkým, keďže má nástroj rozširovať funkčnosť Apache Tika, bolo potrebné zvoliť vhodný spôsob pre jeho napojenie do Tika pracovného postupu. Z možnosti, ktoré sú popísané v kapitole 4.6 sa najlepšie hodí vytvorenie vlastných parserov. Týmto spôsobom sa Tika postará o detekciu typu pomocou *Detector* rozhrania a uľahčí ďalšie spracovanie. Grafickú ukážku napojenie do pracovného postupu Tika je možné vidieť na obrázku 5.3.

Dôležité dáta

Pre správny návrh a následné fungovanie nástroja je nevyhnutná korektná definícia dôležitých dát, na ktoré sa má nástroj zamerať a spracovávať. V prílohe A na obrázku A.1 sú tieto dáta vyznačené červeným rámom na jednom z dodaných dokumentov. Vyznačené časti sú, ako bolo spomínané, tabuľka a dáta typu kľúč/hodnota. Na ďalšom obrázku v prílohe A.2 je možné vidieť niektoré metadáta dokumentu, ktoré budú tiež spracované.

Klasifikačná úloha

Dôležitou súčasťou tohto nástroja bude schopnosť zavolať parser vždy pre dokumenty, pre ktoré bol tento parser implementovaný. MIME typ dokumentu nám už v tomto nepomôže, pretože dokumenty na klasifikovanie sú toho istého typu.

To znamená potrebu správne klasifikovať dokument na základe jeho obsahu a prípadne k tomu využiť aj jeho metadáta. Táto úlohu sa bude riešiť podobne ako Tika rieši detekciu MIME typov pomocou súboru *tika-mimetypes.xml*, v ktorom Tika uchováva záznamy o jednotlivých MIME typoch. Je potrebné vytvoriť súbor v štruktúrovanom jazyku obsahujúci pravidlá pre jednotlivé dokumenty. Konkrétne, pravidlá určujúce aké textové reťazce musí dokument obsahovať aby ho bolo možné klasifikovať ako danú triedu. Ak sa nejaké triedy dokumentov nebudú dať rozlíšiť iba za pomoci ich obsahu, bude potrebné rozšíriť súbor pravidiel o metadáta.

Štrukturalizovaný výstup (JSON) a kontrola jeho validity (JSON Schema)

Aby bol tento nástroj vôbec užitočný, je potrebné aby okrem získania dát, tieto dáta aj spracoval do vhodnej podoby a uložil ich. Pre konzistentný výstupový formát bolo potrebné vybrať jazyk, v ktorom budú výsledné dáta zapísané.

Výber jazyka som obmedzil na množinu jazykov pre polo štruktúrované údaje, ako sú XML, EDI, JSON atď. Z nich som nakoniec ako výsledný jazyk vybral JSON 3.3, pre jeho jednoduchosť, rýchlosť a ľahké zostavovanie v kóde. Na definovanie a overenie štruktúry výstupov bude použitý slovník JSONSchema 3.3.

ReFormator a jeho rozšíriteľnosť

Keďže na vývoj tohto nástroja bola dodaná iba istá sada dokumentov, bolo potrebné pri návrhu zobrať do úvahy možnosť budúceho pridania podpory pre nové dokumenty. To znamená, že by mal nástroj povoľovať pridanie nového parsera a jeho následne zaradenie do procesu práce bez zásahu do existujúceho kódu. Táto potreba sa dá vhodne vyriešiť podobne ako ju rieši Tika.

Teda jednotlivé parsere zodpovedné za extrakciu dát budú implementované ako poskytovatelia služby (SP) 4.6 pre mnou vytvorenú službu *ReParser*. To umožní pridať nový parser rovnakým spôsobom ako keby sme ho pridávali ku Tike 4.6. Jediný rozdiel je v tom, že namiesto *META-INF/services/org.apache.tika.parser.Parser* súboru bude potrebné do JAR súboru okrem skompilovanej triedy vložiť *META-INF/services/re.parsers.service.ReParser* súbor, obsahujúci celé kvalifikované meno nového parsera, ktorý rozširuje *ReParser* a implementuje jeho metódy.

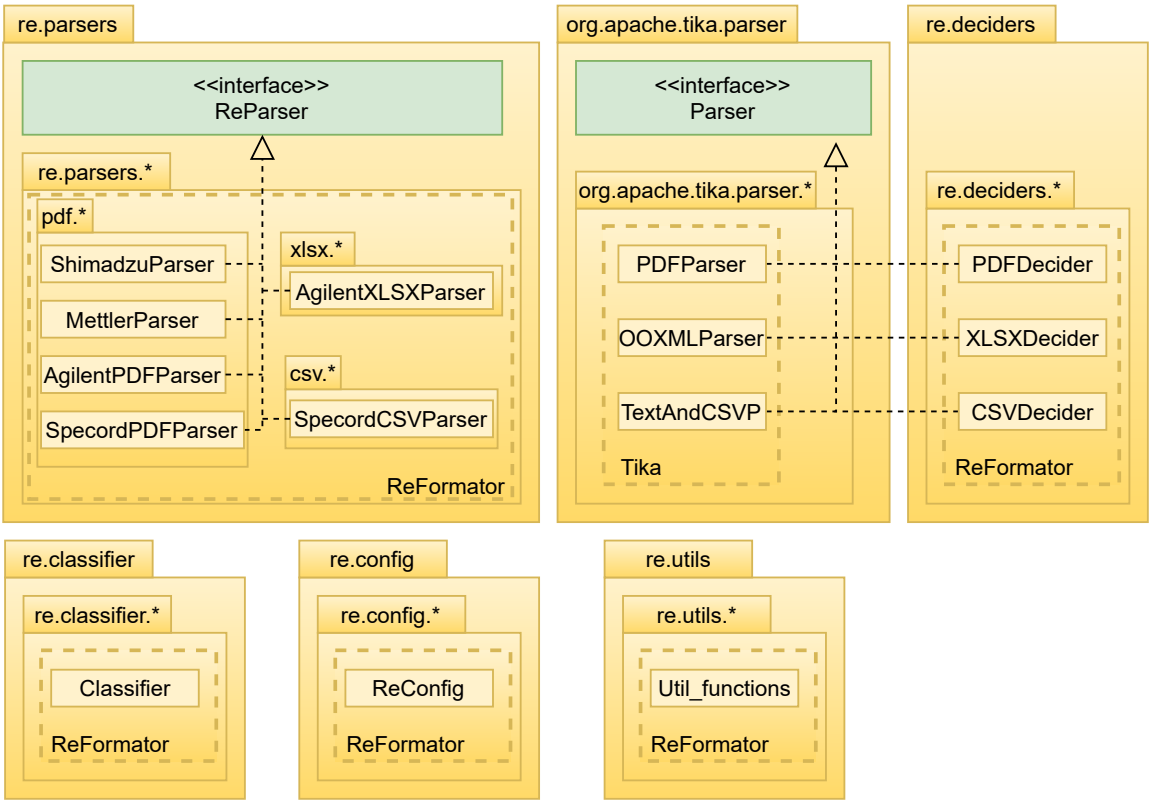
Testovanie

Nástroj by mal byť otestovaný na súboroch dodaných k jeho vývoju, rovnako ako aj na súboroch dodaných v neskoršej fázy vývoja. Otestovaním by sa malo zaistiť jeho správne fungovanie a teda už skôr spomenutá klasifikácia dokumentov, extrakcia dát a validácia výstupu. Tieto testy sa budú robiť vždy pri stavaní projektu pomocou *Maven* nástroja a budú prevedené pomocou *JUint* rámca.

5.3 Architektúra nástroja

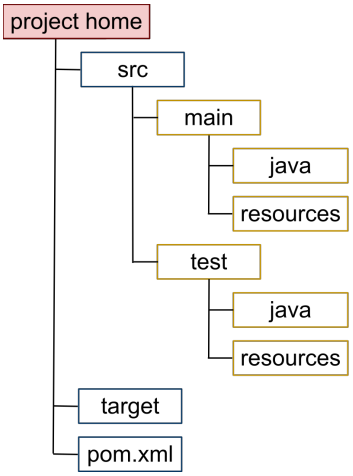
Tento nástroj má jednoduchú funkcionalitu a ani jeho architektúra nie je zložitá. Skladá sa zo štyroch hlavných časti:

- Takzvaní "rozhodujúci" alebo *anglicky Deciders* (v zdrojových súboroch obsahujú slovo *Decider* v názve), ktorí rozširujú AbstractParser rozhranie a plnia funkciu poskytovateľa služieb. Tieto parseri sú prednostne volané Tikou namiesto implicitne nastavených parserov a zabezpečuje prepojenie Tiky s týmto nástrojom. V týchto triedach sa odohráva celý proces inštancovaním ostatných tried a volaním ich metód.
- Trieda *Classifier*, alebo presnejšie metódy, ktoré implementuje sú zodpovedné za správnu klasifikáciu dokumentu pomocou súboru pravidiel. Klasifikovanú triedu posunú ďalej k načítavaniu služieb.
- Rozhranie *ReParser*, ktoré predstavuje službu. Slúži pre načítanie služieb a spojenie s prípadnými rozšíreniami.
- **Poskytovatelia služieb** (jednotlivé implementácia *ReParser* rozhrania). Ich hlavná úloha je parsovanie a extrahovanie dát a taktiež sformovanie, zápis a validácia výstupných dát.
- *Utils* slúži ako pomocná trieda a zhromažďuje všetky potrebné metódy, ktoré využívajú ostatné časti nástroja.



Obr. 5.1: Architektúra jednotlivých prvkov nástroja plus Tika rozhranie *Parser*.

Pre architektúru adresárovej štruktúry 5.2 bola zvolená automaticky generovaná štruktúra nástrojom *Maven*. Tá pozostáva z adresára *src* obsahujúceho zdrojové súbory projektu, adresára *target* kam sa uloží postavený projekt a súboru *pom.xml* obsahujúceho popis konfigurácie projektu pre nástroj *Maven* 6.2.



Obr. 5.2: Hlavná adresárová štruktúra používaná projektami skladanými pomocou *Maven*-u.

5.4 Priebeh spracovania dokumentu

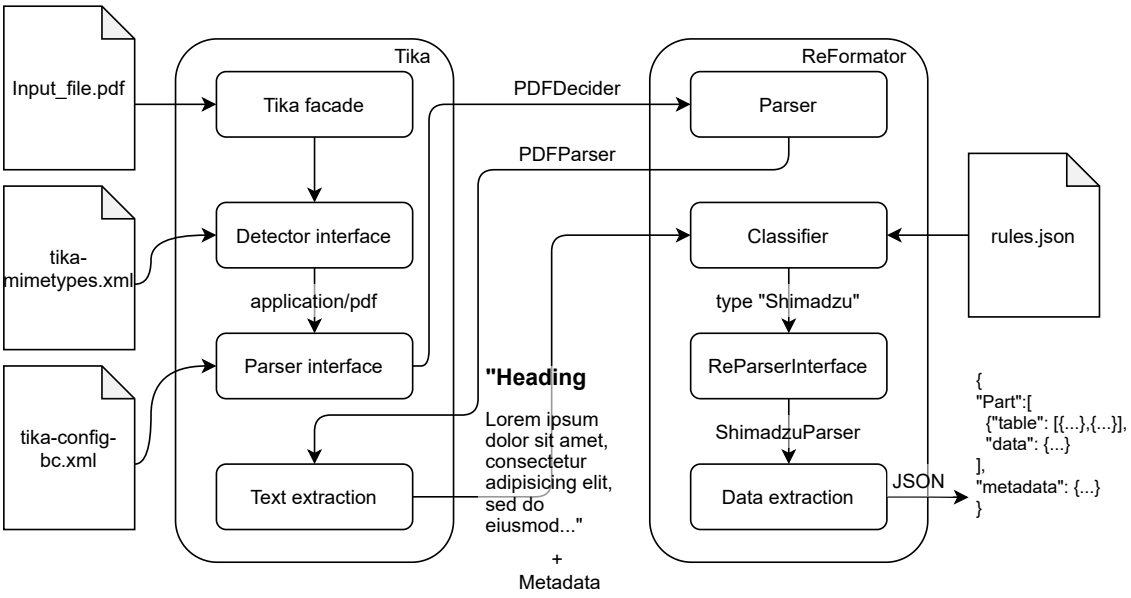
V tejto podkapitole je popísaný pracovný postup 5.3 spracovania dokumentu nástrojom po jeho napojení do Tika pracovného postupu.

1. Ak je dokument takého MIME typu, ktorý ReFormator podporuje (záleží na *tika-config-bc.xml*), predá sa jednému z jeho parserov, zodpovednému za jeho spracovanie (PDFDecider v tomto prípade). Tento "rozhodujúci" parser potom ďalej spracuje dokument pomocou Tika predvoleného parseru (PDFParser) pre typ dokumentu a tým získa jeho obsah a metadáta.

2. Následne *Classifier* využije obsah a metadáta na získanie triedy dokumentu pomocou definovanému súboru pravidiel *rules.json*.

3. Ďalej je pomocou tejto triedy načítaný príslušný poskytovateľ služieb za pomoci *ReParser* rozhrania poskytovateľa služieb.

4. Poskytovateľ dokument spracuje, extrahuje z neho dôležité informácie, poskladá ich podľa určenej schémy, zapíše do výstupného súboru a nakoniec validuje výsledok oproti schéme pre výstupné dáta.



Obr. 5.3: ReFormator workflow po napojení do Tika workflow

Kapitola 6

Implementácia

Táto kapitola popisuje konkrétne implementačné detaily nástroja ReFormator. Najprv sa zameriava na to, ako bol nástroj integrovaný 6.2 pomocou JAR súborov. Potom popisuje klasifikáciu dokumentov 6.3 a implementáciu jednotlivých vlastných parserov 6.4. Nakoniec rozoberie tvorbu výstupných JSON dát a ich overenie pomocou JSONSchema slovníku 6.6.

6.1 Použité technológie

Celý nástroj je implementovaný v jazyku Java[1], konkrétne Java 15. Na testovanie sa použil JUnit 4 (pozri 7.1) a v neskoršej fáze aj *bash*[2], v ktorom boli napísané aj niektoré pomocné skripty. Vývoj prebiehal v IntelliJ IDEA Ultimate Edition[3] a výsledný nástroj je zostavený pomocou *Maven* nástroja.

6.2 Integrácia pomocou JAR súborov

Formát súboru Java ™ Archive (JAR) [11][12] umožňuje zoskupiť viac súborov do jedného archívneho súboru na distribúciu. Súbor JAR zvyčajne obsahuje súbory Java tried a pomocné prostriedky ako súvisiace metadáta a zdroje (text, obrázky, dokumenty atď.). Súbory tohto typu sú postavené na formáte ZIP 3.2, takže ich je možné použiť na úlohy, ako je napríklad bezstratová kompresia dát, archivácia, dekompresia a rozbalenie archívu. Okrem týchto základných funkcií ponúkajú aj iné, pokročilé funkcie ako napríklad elektronický podpis. Tieto súbory majú zvyčajne príponu *.jar* a z technického hľadiska nie sú ničím iným ako premenovanými ZIP archívmi. Možno ich teda vytvoriť pomocou ľubovoľného ZIP nástroja. Bežnejšie sa však používa príkaz:

```
$ jar cf jar-file input-file(s)
```

Tento príkaz vygeneruje skomprimovaný JAR súbor a uloží ho do aktuálneho adresára. Príkaz taktiež vytvorí základný MANIFEST súbor pre JAR archív.

MANIFEST [11] je špeciálny súbor, ktorý môže obsahovať informácie o súboroch zabalených v súbore JAR. Prispôsobením "meta" informácií, ktoré obsahuje manifest, sa súboru JAR umožňuje slúžiť na rôzne účely. Niektoré z najdôležitejších informácií, ktoré súbor MANIFEST.MF obsahuje sú:

[1] `https://www.java.com/en/`

[2] Bash je unixový príkazový shell interpreter naprogramovaný v rámci projektu GNU. Názov je akronym k názvu Bourne again shell – je založený na Bourne Shellu, čo bol najpoužívanejší unixový shell [7].

[3] `https://www.jetbrains.com/idea/`

- verzia manifest špecifikácie a názov tvorcu

```
Manifest-Version: 1.0
Created-By: 1.7.0_06 (Oracle Corporation)
```

- odkazy na iné súbory JAR, ktoré slúžia ako pomocné programy na účely tejto aplikácie[4]. Hlavička Class-Path má nasledujúcu formu:

```
Class-Path: jar1-name jar2-name directory-name/jar3-name
```

- vstupný bod aplikácie, teda ktorú triedu spustiť ako *Main-Class*.

```
Main-Class: classname
```

  Vstupný bod je trieda obsahujúca metódu s nasledujúcim popisom:

```
public static void main(String[] args)
```

  Po nastavení hlavnej triedy v manifeste je možné aplikáciu spustiť *java* príkazom, ktorý vykoná *main* metódu triedy uvedenej ako *Main-Class* v hlavičke:

```
java -jar JAR-name
```

Maven

Aby som nemusel nastavovať všetky hodnoty MANIFEST súboru ručne, použil som nástroj *Maven* [15], ktorý zostavuje projekt pomocou svojho **projektového objektového modelu (POM)** a sady doplnkov a výrazne šetrí čas pri navigácii v mnohých projektoch. Aj keď použitie *Maven*-u nevylučuje potrebu poznať základné mechanizmy, ušetrí vývojárov od mnohých detailov.

POM (Project Object Model) poskytuje všeobecnú konfiguráciu pre jeden projekt. Tá pokrýva názov projektu, jeho vlastníka a jeho závislosti od iných projektov. Väčšie projekty (napr. Apache Tika) by mali byť rozdelené do niekoľkých modulov alebo podprojektov, každý s vlastným POM. Jeden potom napíše koreňový POM, cez ktorý môže zostaviť všetky moduly pomocou jediného príkazu. POM je uložený v *pom.xml* súbore, ktorý môže vyzerať podobne ako ukážka jeho časti z môjho projektu na výpise 2:

Z obsahu tohto súboru je známe, že výsledný JAR archív sa bude volať "ReFormat-1.0-SNAPSHOT" a že okrem iných archívov (ktoré sú na ukážke nahradené "..." medzi *<dependency>* a *</dependencies>*) je závislý na *tika-app-2.0.0-SNAPSHOT.jar*. Pre vybudovanie projektu a jeho komprimáciu do JAR súboru je potrebné sa uistiť súbor *pom.xml* umiestniť do domovského adresára projektu a následne z tohto adresára zavolať príkaz v príkazovom riadku a nechať všetko na *Maven*:

```
$ mvn package
```

Zavolaním tohto príkazu sa vykonajú všetky fázy[5] až po fázu napísanú v príkaze, teda *package*. *Maven* okrem iných pozná (v uvedenom poradí) tieto hlavné fázy: *validate, compile, test, package, verify, install* a *deploy*. Pre tento príkaz teda prebehnú fázy *validate, compile, test* a *package*.

Výslednú aplikáciu by sme spustili takto:

[4]Triedy sa určujú, aby sa zahrnuli do poľa hlavičky Class-Path v súbore manifestu aplikácie. Použitím hlavičky Class-Path v manifeste je možné sa vyhnúť nutnosti zadávať dlhý príznak -classpath pri vyvolaní Javy na spustenie aplikácie.

[5]Fáza je krok v životnom cykle zostavenia, kde zostavenie je usporiadaná postupnosť fáz

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <modelVersion>4.0.0</modelVersion>
    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.apache.tika</groupId>
            <artifactId>tika-app</artifactId>
            <version>2.0.0-SNAPSHOT</version>
        <dependency>...
    </dependencies>
    <build></build>
    <groupId>org.parsers</groupId>
    <artifactId>ReFormat</artifactId>
    <version>1.0-SNAPSHOT</version>
</project>
```

Výpis 2: Časť súboru pom.xml z tohto projektu

```
$ java -jar <path-to>/ReFormat-1.0-SNAPSHOT.jar <file_to_parse>
```

Tento archív však obsahuje iba zdrojové súbory a dáta z projektového adresára. Neobsahuje žiadne projekty, na ktorých je závislý. Preto by bolo nutné JAR súbory týchto projektov umiestniť do adresára rovnakého adresára ako je tento archív, alebo by ich bolo nutné pridať s celou cestou do príkazu nasledovne:

```
$ java -jar <path-to>/ReFormat-1.0-SNAPSHOT.jar:<path-to>dependency_1.jar
    ...:dependency_N.jar Main-Class
```

Aby som nemusel písať zbytočne dlhý príkaz alebo zhromaždovať všetky archívy v jednom adresári, pridal som do *pom.xml*, konkrétne do sekcie *<build></build>* doplnok *maven-assembly-plugin* viditeľný na výpise 3, ktorý vytvorí okrem predošlého archívu ešte jeden obsahujúci všetky závislosti. Tento archív je možné spustiť bez ďalších dodatočných súborov. Okrem toho som mu zadal implicitnú hlavnú triedu *org.apache.tika.cli.TikaCLI* a názov explicitne zmenil z *ReFormat-jar-with-dependencies.jar* na *ReFormat.jar*. Nakoniec je vďaka časti *execution* možné postaviť iba tento samostatne fungujúci archív a to použitím príkazu:

```
$ mvn (clean) compile assembly:single
```

Po tejto úprave by sa už aplikácia spúšťala takto:

```
$ java -jar <path-to>/ReFormat.jar <file_to_parse>
```

Pre žiadanú funkčnosť však je ešte potrebné pridať poslednú úpravu k tomuto príkazu a to je zadanie konfiguračného súboru pre Tiku, ktorý nahradí Tika implicitné parsere mojimi vlastnými. Vďaka tomu, že som ako hlavnú triedu uviedol *TikaCLI*, je možné do príkazu

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>3.3.0</version>
    <configuration>
        <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
        <finalName>ReFormat</finalName>
        <appendAssemblyId>false</appendAssemblyId>
        <attach>false</attach>
        <archive>
            <manifest>
                <addClasspath>true</addClasspath>
                <mainClass>org.apache.tika.cli.TikaCLI</mainClass>
            </manifest>
        </archive>
    </configuration>
    <executions>
        <execution>
            <id>assemble-all</id>
            <phase>package</phase>
            <goals>
                <goal>single</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Výpis 3: Plugin pridaný do pom.xml

pridať akýkoľvek argument ktorý Tika pozná a bude spracovaný ako keby som spúšťal Tiku. Použiteľný a výsledný príkaz nakoniec vyzerá takto:

```
$ java -jar <path-to>/ReFormat.jar --config=<path-to>/<tika-config-bc.xm>
<file_to_parse>
```

6.3 Implementácia klasifikácie jednotlivých dokumentov

Na rozpoznávanie MIME typov využíva ReFormator *Detector* rozhranie od Apache Tika. Na rozoznávanie a klasifikáciu jednotlivých tried súborov rovnakého MIME typu, bolo treba implementovať vlastný klasifikátor. Môj prvý pokus implementácie využíval Weka rozhodovací strom.

Weka klasifikátor

Weka [24] je softvér s otvoreným zdrojovým kódom[6], ktorý poskytuje nástroje na predspracovanie údajov, implementáciu niekoľkých algoritmov strojového učenia a vizualizačné nástroje, na vývin techniky strojového učenia a ich aplikácie na problémy s dolovaním údajov v reálnom svete.

Weka modely pracujú s údajmi zapísanými vo formáte *.csv* a *.arff* 4. Tieto modely sa dajú dopredu natrénovať a uložiť pre neskoršie načítanie. Prvý klasifikátor mal dokumenty rozoznávať iba na základe ich metadát, čo sa hneď ukázalo ako nie veľmi presné, kvôli veľkej rôznorodosti jednotlivých hodnôt metadát (Rôzne klasifikačné triedy, ktoré si vyžadovali iný prístup obsahovali rovnaké metadáta).

Druhý model už využíval textový obsah dokumentu, v ktorom vyhľadával reťazce typické pre konkrétne triedy a riadil sa podľa hodnôt: *true* ak reťazec obsahuje a *false* ak nie. Z týchto hodnôt vytvoril inštanciu pre klasifikovanie. Týmto prístupom som už získal väčšiu kontrolu nad rozdeľovaním do tried, avšak aj napriek tomu nie dostatočnú. Táto metóda sa taktiež ukázala byť zložitá pre rozšírenie, keďže pre každý nový typ s novými typickými reťazcami by bolo treba rozšíriť aj predošlé typy a zisťovať či tieto reťazce náhodou neobsahujú.

```
@relation file-contains

@attribute description  {true, false}
@attribute status  {true, false}
@attribute content  {true, false}
@attribute sample_id  {true, false}
@attribute designation  {true, false}
@attribute meas_mode  {true, false}
@attribute meas_values  {true, false}
@attribute rt_min_rt  {true, false}
@attribute sample_description  {true, false}
@attribute column_max_temp  {true, false}
@attribute parser        {none, ShimadzuParser, AgilentParser,
↪  SpecordPDFParser, MettlerParser}
@data
% 4 instances
true,true,false,true,false,false,false,false,false,true,ShimadzuParser
false,false,true,true,false,false,false,false,false,false,MettlerParser
false,false,false,false,true,true,true,false,false,false,SpecordPDFParser
false,false,false,false,false,false,false,false,true,false,AgilentParser
```

Výpis 4: Obsah súboru rules.arff použitý pre trénovanie Weka modelu na klasifikáciu pomocou obsahu dokumentov

[6] `https://svn.cms.waikato.ac.nz/svn/weka/branches/stable-3-8/`

```
sample_id = true
|  description = true: ShimadzuParser
|  description = false: MettlerParser
sample_id = false
|  designation = true: SpecordPDFParser
|  designation = false: AgilentParser
```

Výpis 5: Rozhodovací strom Weka modelu a jeden z dôvodov prečo bol nakoniec použitý iný spôsob klasifikácie, keďže rozhodnutie nakoniec záležalo iba na troch prvkoch (*sample_id, description, designation*).

Nakoniec som sa rozhodol nepoužiť Weka klasifikátor, z dôvodu jeho nekompatibility s danou klasifikačnou úlohou a jej zbytočnom komplikovaní pri jeho použití.

Súbor pravidiel

Namiesto využitia existujúcich klasifikačných stromov a nástrojov som sa rozhodol implementovať vlastný rozhodovací systém. Tento systém by sa riadil externým súborom pravidiel, ktorý by sa dal jednoducho rozšíriť o nové klasifikačné triedy. Okrem toho by mal dodržovať istú schému, oproti ktorej by bol kontrolovaný pre prípadné chybné zmeny alebo rozšírenia súboru.

Prvá verzia tohto súboru, ktorej schémy som sa držal až do finálnej verzie bola napísaná v jazyku XML a vyzerala následovne:

```
<?xml version="1.0"?>
<decision_tree>
    <pdf>
        <parser id="Shimadzu">
            <text>
                <element value="Injection Volume"/>
                ...
            </text>
            <metadata>
                <element key="pdf:producer">
                    <value value="SkyPDF Server MT 2016 Version 6.0.2..."/>
                    <value value="GPL Ghostscript 8.70"/>
                    ...
            </metadata>
        </parser> ...
```

Výpis 6: Časť súboru rules.xml

Ako je na výpise 6 vidieť, všetky klasifikačné triedy jedného MIME typu sa nachádzajú v elemente s názvom odvodeným od tohto MIME typu. Názov klasifikačnej triedy je ľubovoľný reťazec, ktorý potom parser implementovaný pre túto triedu použije ako svoj identifikátor pri jeho načítaní. Tento reťazec je prítomný v elemente *<parser>* ako atribút *id* (v tomto prípade "Shimadzu"). Parser element obsahuje dva pod-elementy:

- *<text>* element, ktorý obsahuje elementy s atribútom *value*, ktorého hodnotu musí textový formát dokumentu obsahovať aby mohol byť klasifikovaný ako konkrétna trieda.
- *<metadata>* element, obsahujúci okrem *value* aj atribút *key*, ktorého hodnota slúži ako kľúč pre vyhľadávanie v metadátach. Pre jeden kľúč, môže byť na vyhľadanie viacero hodnôt.

JSON vs. XML

Pre porovnanie som skúsil napísať súbor pravidiel podľa rovnakej schémy v jazyku JSON 3.3. Okrem toho, že tento súbor bol čitateľnejší viď. 7, bola implementácia jeho prechádzania a použitia na klasifikáciu oveľa jednoduchšia.

```
{
  "pdf": {
    "Mettler": {
      "text": ["Name", "Sample ID", "Sample size", "Content R1", "State",
                "TitrationReport", "Titration", "Scope", "Curve"],
      "metadata": {
        "dc:title": ["TitrationReport"],
        "pdf:docinfo:producer": "Microsoft: Print To PDF"
      }
    },
```

Výpis 7: Ukážka zo súboru pravidiel rules.json

Pre rozhodnutie medzi týmito dvoma súbormi a a jazykmi som napísal krátky test, v ktorom mali za úlohu 100 000 - krát klasifikovať rovnaký súbor. Výsledky potvrdili to čo tvrdí mnoho zdrojov na internete[7], XML je zložitejší na parsovanie, čo ho činí pomalším:

```
Priemer časov z 5-ich behov:

XML pre 100 000 rozhodnutí
37.038 s

JSON pre 100 000 rozhodnutí
5.188 s

XML priemerne na jedno rozhodnutie: 0.37038 ms
JSON priemerne na jedno rozhodnutie: 0.05188 ms
Moja implementácia s JSON je priemerne 7.139-krát rýchlejšia ako moja
↪  implementácia s XML
```

Výpis 8: Výpis konzoly pre krátky test rýchlosti klasifikácie medzi súbormi pravidiel v JSON a XML formátoch

[7] https://www.w3schools.com/js/js_json_xml.asp

Konečná klasifikačná metóda teda využíva súbor pravidiel napísaný v JSON formáte a jej zavolanie je možné vidieť na výpise 9. Najprv je nutné vytvoriť inštanciu *Classifier* triedy, kde argument *mime* určuje o aký MIME typ dokumentu ide a v ktorej časti súboru pravidiel sa má vytvorený klasifikátor pohybovať ("pdf" pre PDF súbory, "xlsx" pre XLSX súbory,...). Argument *rules* predáva JSON objekt obsahujúci pravidlá, podľa ktorých má klasifikátor rozhodovať. Tieto parametre konštruktéra je možné zmeniť metódami *setRules(JSONObject)* a *setType(String)*.

Vo volaní metódy by mal potom argument *textContent* obsahovať textový obsah súboru na klasifikovanie a *metadata* by mali byť metadáta získane prvotným parsovaním dokumentu niektorým z Tika parserov. Táto metóda vracia reťazec predstavujúci meno klasifikačnej triedy (napr. "Shimadzu"), pre zhodu a *null* ak dokument nepatrí k žiadnej známej klasifikačnej triede.

```
Classifier classifier = new Classifier(mime, rules);
classifier.decideJSON(textContent, metadata);
```

Výpis 9: Príklad volania metódy tried *Classifier* na klasifikovanie

Získavanie hodnôt pre súbor pravidiel

Hodnoty, ktoré sú v súbore som pre prvotne dodané dokumenty (bolo ich menej ako 10 pre jednu triedu) hľadal a získaval ručne. Po obdržaní ďalších dokumentov (tentokrát okolo 100) som však si však potreboval pomôcť. Napísal som pomocnú metódu, využívajúcu Tika textové extrahovanie. Výsledný text som v metóde rozdelil pomocou regulárnych výrazov na frázy oddelené novým riadkom. Túto metódu som zavolal v *Main* triede a pomocou tohto skriptu získal reťazce opakujúce sa v dokumentoch rovnakej triedy:

```
FILES="/home/rene/Desktop/Bakalarka/test_folder/Shimadzu/*"
for f in $FILES
do
  echo "Processing $f file..."
  ./parse.sh "$f" > 1  && comm -12 <(uniq 1 | sort 1) <(uniq 2 | sort 2) >
  ↪  3 && cat 3 > 2
  done
uniq 2 > fin
```

Výpis 10: Obsah krátkeho skriptu *forfiles.sh* použitého k získavaniu rovnakých reťazcov z dokumentov patriacich k rovnakej klasifikačnej triede

Pre metadáta som napísal jednoduchú metódu na ich výpis. Potom som našiel všetky hodnoty, ktoré môže nadobudnúť daný kľúč a zapísal ich do pravidiel vo forme poľa reťazcov.

Tento prístup bol dostačujúci pre malý počet klasifikačných tried a umožnil mi nájsť dostatok dát pre úspešnú klasifikáciu. Do budúcnosti by však bolo určite lepšie prísť s kvalitnejšou metódou na získavanie dát pre pravidlá.

6.4 Implementácia jednotlivých parserov

Pre každú triedu klasifikácie bolo potrebné implementovať aj vlastný parser, ktorý by s ohľadom na špecifickosť dokumentu vyhľadal potrebné dáta. Parsere sú rozdelené podľa MIME typov dokumentov s ktorými pracujú. Pre obe XLSX 6.4 a CSV 6.4 MIME typy súborov postačovala jedna klasifikačná trieda pre dodané súbory jedna klasifikačná trieda a teda jeden parser, avšak súbory typu PDF 3.2 boli rozdelené do štyroch tried a každá si vyžadovala špecifický prístup.

Každý z týchto parserov rozširuje *ReParser* rozhranie a implementuje jeho 2 metódy: *parse()*, zodpovednú za extrahovanie dát a *getType()*, ktorá vracia meno triedy a je využívaná pri volaní načítaní poskytovateľa služby.

CSV

CSV súbory [19] využívajú jednoduché formátovanie. Súbory sú vo forme čistého textu a pozostávajú zo záznamov oddelených novým riadkom ("`\n`"). Tieto záznamy sú v jednom súbore rozdelené rovnakým znakom (",", ";", alebo pre TSV "`\t`") a zvyčajne majú rovnaký počet stĺpcov. Prvý riadok sa zvyčajne líši od zbytku dokumentu a určuje názvy jednotlivých stĺpcov. Tieto dokumenty majú zvyčajne veľký počet riadkov.

Tieto dokumenty patria do mnou vymyslenej klasifikačnej triedy "SpecordCSV" a k ich parsovaniu bola vytvorená *SpecordCSVParser* Java trieda. V nej som dáta získaval z textového obsah dokumentu, získaného pomocou *TextAndCSVParser* triedy v Tike. Stačilo iterovať cez riadku a deliť ich pomocou deliaceho znaku, ktorý tieto súbory využívajú (";").

PDF

PDF dokumenty predstavujú jeden z najhorších MIME typov na extrahovanie a štrukturalizáciu dát. Aj keď sa na prvý pohľad môžu zdať čitateľné a pekne sformované 3.2, je to preto, že sú za týmto účelom tvorené. Rozdiel však nastáva ak sa ich snaží prečítať počítač. Ako je názorne vidieť na výpise 11, ktorý reprezentuje extrahovaný text z časti dokumentu na obrázku 6.1. Dáta môžu byť rôzne poprehadzované či už sprava-doľava, alebo zhoranadol. Dôležité je aj v takto pomiešaných dátach nájsť súvislosti a pravidlá, ktoré by sa dali aplikovať pre celú triedu súborov v takejto forme.

Dodané PDF súbory som rozdelil do štyroch klasifikačných tried: "AgilentPDF", "Mettler", "Shimadzu" a "SpecordPDF". Pre každú triedu som implementoval jeden parser zameraný na extrahovanie hodnôt podľa kľúčových slov špecifických pre danú triedu a to taktiež spôsobom pre ňu prispôsobeným. Tieto spôsoby sa odvíjali napríklad od postupností dát, použitého delitľa dát alebo spomínaných kľúčových slov.

```
StateResultNameSample ID
0,8033 gSample size105696, C/10/20-MEOH21

OK0,0793 %Content R1
0,7373 gSample size105696, C/10/20-MEOH22

OK0,0714 %Content R1
```

Výpis 11: Výstup po extrahovaní z časti dokumentu na obrázku 6.1

	Sample ID	Name	Result	State
1	105696, C/10/20-MEOH2	Sample size	0,8033 g	
		Content R1	0,0793 %	OK
2	105696, C/10/20-MEOH2	Sample size	0,7373 g	
		Content R1	0,0714 %	OK

Obr. 6.1: Časť pdf súboru, z ktorej bol extrahovaný výpis 11 pomocou Tika *PDFParser*

Tento súbor patrí pod klasifikačnú triedu "Mettler", ktorú parsuje *MettlerParser*.

XLSX

XLSX [8] je tabuľkový formát súborov, ktorý je podtypom z rady typov Microsoft Office (konkrétne Microsoft Excel) a používa tabuľky na organizáciu, analýzu a uloženie dát. Jeho celé meno je "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet".

Narozdiel od predošlých parserov som pre súbory typu XLSX ako pre jediné pri parsovaní nepoužil ich textový obsah vo formáte *String*, ale spracovával ich pomocou *InputStream*-u a triedy XSSFWorkbook(). Táto trieda mi dovoľuje previesť *InputStream* na objekt, pomocou ktorého iterátora môžem prechádzať XLSX dokumenty bunku po bunke. To výrazne uľahčilo dostať sa ku konštantným a celistvejším výsledkom parsovania.

Problém pri tomto prístupe bol však samotný *InputStream*, ktorý je možné čítať len raz. Program ho prvý krát číta pri získavaní textového obsahu pre klasifikátor pomocou Tika *OOXMLParser*. Potom ho potrebuje čítať pre samotnú extrakciu dát.

Štandardným prístupom [16] je poznačiť si *InputStream* pred jeho prečítaním pomocou *mark(int)* a následne ho po jeho prečítaní "reštartovať" pomocou *reset()*. *InputStream* predaný od Tiky však tieto operácie nepodporuje a tak som tento problém musel vyriešiť inak. To som urobil duplikovaním *InputStreamu*[8], ako je možné vidieť na výpise 12. Jeden tok po tomto využije Tika *OOXMLParser* a druhý môj *AgilentXLSXParser*.

```
byte[] buffer = new byte[16000];
ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
int len;
while ((len = inputStream.read(buffer)) > -1) {
    outputStream.write(buffer, 0, len);
}
outputStream.flush();

InputStream in1 = new ByteArrayInputStream(outputStream.toByteArray());
InputStream in2 = new ByteArrayInputStream(outputStream.toByteArray());
```

Výpis 12: Kúsok kódu, ktorý urobí z jedného *InputStream*-u dva. Prevzatý z [16]

6.5 Service Provider

Ako bolo spomínané v časti 5.2, pre rozšíriteľnosť sú parseri implementované ako poskytovatelia služby. No okrem výhody ľahkého rozšírenia, poskytuje táto implementácia aj

[8] https://developpaper.com//how-to-reuse-inputstream/

jednoduchý spôsob ich načítavania. K tomu sa využíva *ServiceLoader* [17] ako je vidieť na prvom riadku výpisu 14.

Týmto príkazom sa do premennej *loader* načítajú všetci poskytovatelia služieb, ktorí sú na CLASSPATH (a ich absolútne meno triedy je zapísané v súbore *META-INF.services/re.parser.service.ReParser*). Cez *loader* premennú sa potom iteruje a hľadá sa v nej poskytovateľ, ktorý po zavolaní jeho implementácie metódy *getType()* vráti rovnaké meno triedy aké vrátil klasifikátor.

Test rýchlosti iterovania

Pre výber najlepšieho spôsobu iterácie som znova napísal krátky rýchlostný test. Jeho výsledky je možné vidieť na výpise 13. Najrýchlejší prístup pomocou iterátora bol viac ako 9000-krát rýchlejší než prístup na poslednom mieste a približne 60-krát rýchlejší než implementácia na druhom mieste. Na výpise 14 je potom možné vidieť výslednú implementáciu pomocou najrýchlejšieho prístupu.

```
Pre 100 000 iterácií:
45.7984 sekúnd pre loader.forEach()
51.9229 sekúnd pre for(ServiceProvider sp : loader)
0.0057  sekúnd pre iter = loader.iterator => while(iter.hasNext())
0.3366  sekúnd pre loader.stream().filter(l -> l.toString().equals(type))
```

Výpis 13: Výpis pre test iterovania cez ServiceLoader. Na ľavej strane je čas pre daný prístup a na pravej krátky popis iterovacieho prístupu

```
ServiceLoader<ReParser> loader = ServiceLoader.load(ReParser.class);
Iterator<ReParser> iter = loader.iterator();
while(iter.hasNext()){
    try {
        ReParser p = iter.next();
        if (p.getType().equals(type)) {
            p.parse(inputStream, handler, metadata);
            validateOutput(metadata, reConfig);
            break;
        }
    }
    catch (IOException | TikaException | SAXException |
    ↪  NoSuchElementException e) {
        e.printStackTrace();
    }
}
```

Výpis 14: Výsledné načítavanie poskytovateľa služieb

6.6 Výstupné JSON dáta

Pre tvorbu výstupných JSON dát som použil metódy triedy com.google.gson.JsonObject[9]. Tie mi dovolili tvoriť vnorené JSON objekty a polia. V niektorých parseroch, ak to štruktúra textových dát dokumentu dovolí, je JSON výstup skladaný priamo pri extrakcii dát. Pre ďalšie triedy dokumentu bolo lepšie si tieto dáta uložiť do pomocných premenných a poskladať ich na konci. Na konkrétnej implementácií nie je čo popisovať, keďže sa jedná iba o klasické mapovanie hodnôt na kľúče do JSON objektov a polí a ich následné vnorenie do nadradených objektov.

Výstupné dáta sú rozdelené do dvoch JSON objektov. Prvý s kľúčom "metadata" obsahuje všetky metadáta získané v priebehu parsovanie vo forme kľúč/hodnota. Druhý s kľúčom "content" obsahuje pole JSON objektov, kde každý objekt predstavuje časť dokumentu (meranie, stranu,...).

Tento objekt potom obsahuje kľúč "data", v ktorom sú uložené nájdené dôležité dáta z konkrétnej časti vo formáte kľúč/hodnota. Kľúč "table_#" je prítomný, ak dokument obsahoval aspoň jednu tabuľku (štruktúra objektu tabuľky závisí na konkrétnom dokumente). Nepovinný kľúč "section_name" obsahuje pomenovanie sekcie dokumentu (napr. "Page 1", "Analysis 2",...).

```
import json

f = open("output.json")
data = json.load(f)
metadata = data["metadata"]
pdf_version = data["metadata"]["pdf:PDFVersion"]
section_object = data["content"][0]
table_object = data["content"][0]["table"]
table_row_3 = data["content"][0]["table"]["3"]
data_object = data["content"][0]["data"]
```

Výpis 15: Príklad prístupu k vygenerovaným dátam v Pythone

Čitateľný výstup vďaka GsonBuilder

Na výpise 16, je možné vidieť ako vyzerali výstupné dáta. Aj keď takýmto dátam počítač stále dokonalo rozumie, pre človeka sú už skoro nečitateľné a nie je možné si vybaviť hierarchiu dát (plus v nie každom textovom editore sú hodnoty a kľúče farebne odlíšené). Preto tento výstup ešte upravujem pomocou metódy vo výpise 18. Je to krátka a jednoduchá úprava, ale je to oveľa príjemnejšie ak dáta vyzerajú ako ukážkové výstupné dáta na výpise 17, najmä ak ich budú čítať ľudia.

[9] https://www.javadoc.io/doc/com.google.code.gson/gson/2.8.5/com/google/gson/JsonObject.html

```
{"content":[{"section_name":"Report 1","data":{"Project
↪  Name":"4-FBSNa","Instrument":"QC-HPLC-2","Sample name":"4-FBSNa
↪  349/120","Column name":"Symmetry Shield RP8 (ev.č.632)","LIMS ID":"OA
↪  107721","Serial #":"","Sequence Name":"2020-05-28c","Diameter":"4.60
↪  mm","Acq. method":"4-FBSNa_CD.amx","Length":"100.0 m","Processing
↪  method":"FBSNa","Particle size":"","Injection date":"05/28/2020
↪  14:27:51","Location":"Vial 4","Operator":"Katerina Prchalova","Inj.
↪  volume":"10.0","Processed by":"Katerina Prchalova"
```

Výpis 16: Prvotne poskladané JSON dáta

```
{
  "content": [
    {
      "section_name": "Report 1",
      "data": {
        "Project Name": "4-FBSNa",
        "Instrument": "QC-HPLC-2",
        "Sample name": "4-FBSNa 349/120",
        "Column name": "Symmetry Shield RP8 (ev.č.632)",
        "LIMS ID": "OA 107721",
        "Serial #": "",
        "Sequence Name": "2020-05-28c",
        "Diameter": "4.60  mm",
        "Acq. method": "4-FBSNa_CD.amx",
        "Length": "100.0 m",
        "Processing method": "FBSNa",
        "Particle size": "",
        "Injection date": "05/28/2020 14:27:51",
        "Location": "Vial 4",
        "Operator": "Katerina Prchalova"
```

Výpis 17: Úpravené JSON dáta

```
private static String prettify(JsonObject jsonObject) {
    Gson gson = new GsonBuilder().setPrettyPrinting().create();
    JsonParser jp = new JsonParser();
    JsonElement je = jp.parse(jsonObject.toString());
    String prettyJsonString = gson.toJson(je);
    prettyJsonString =
    ↪  org.apache.commons.text.StringEscapeUtils.unescapeJava(prettyJsonString);
    return prettyJsonString;
}
```

Výpis 18: Metóda *prettify(JsonObject)* volaná pre všetky výstupné dáta.

Validácia JSON dát

Všetky JSON dáta v tomto projekte majú definovanú štruktúru pomocou JSON Schema slovníka a sú ňou aj následne validované. To platí pre výstupy všetkých implementovaných parserov aj pre súbor pravidiel *rules.json*. Tieto schémy sú v zdrojových súboroch pod názvami *schema_output.json* pre výstupy a *schema_rules.json* pre súbor pravidiel.

V prílohe C je možné vidieť príklad výsledných dát, validných oproti schéme *schema_output.json*, ktorú je možné vidieť v prílohe B. Dáta v prílohách boli extrahované z dokumentu v prílohe A, ktorý patrí do klasifikačnej triedy "AgilentPDF".

- Dáta musia obsahovať jedno JSON pole objektov "content" a jeden JSON objekt "metadata".
- Objekt "metadata", už nemôže obsahovať ďalšie objekty ani polia, ale iba základné typy[10]. Ako jeho názov naznačuje, obsahuje metadáta dokumentu.
- Pole "content" obsahuje JSON objekty, ktoré predstavujú jednu časť dokumentu (meranie, stranu, analýzu,...).
- Každý objekt v poli "content" môže obsahovať kľúče "section_name", "table_#" a musí obsahovať kľúč "data".
- Hodnota "section_name" predstavuje názov časti dokumentu (napr. Page 1, Analysis 2,...) a musí to byť reťazec.
- Hodnota "table_#" predstavuje tabuľku a musí to byť JSON objekt. Tento objekt už nie je nijako obmedzovaný.
- Hodnota "data" predstavuje všetky ostatné dôležité dáta typu meno/hodnota nájdené v dokumente. Všetky hodnoty musia byť základného typu.

Štruktúra výstupných dát je kontrolovaná vždy po preparsovaní dokumentu a ich zápise do súboru pomocou metódy *validateOutput(metadata, reConfig)* ako je možné vidieť na výpise 14. Atribúty tejto metódy sú *metadata* daného dokumentu, pretože z nich je vyvodené výsledné meno JSON súboru a *reConfig* 6.7, z ktorého načíta schému ku validácií. Podobne je validovaný súbor *rules.json* v "rozhodovacích" parseroch vždy pred klasifikáciou.

[10] reťazce, čísla, pravdivostné hodnoty alebo hodnota *null*

```
{
  "content": [
    {
      "section_name": "Page 1",
      "data": {"Project Name": "MBS"},
      "table": {
        "Signal": "DAD1A,Sig=258,4  Ref=off",
        "1": {"RT [min]": "2.351"},
        "2": {"RT [min]": "4.376"},
        "Area Sum": "19764.29"}
    },
    {
      "section_name": "Page 2",
      "data": {"Project Name": "String value"}
    }
  ],
  "metadata": {"protected": "false"}
}
```

Výpis 19: Ukážka štruktúry výstupných dát. (Bez niektorých odsadení pre skrátenie výpisu.)

6.7 ReConfig

ReFormator by bol v tomto momente pripravený na použitie. Ale ak by niekto chcel zmeniť adresár pre ukladanie výstupných súborov, alebo nahradiť súbor pravidiel novým, pričom by chcel starý ponechať nezmenený. K tomuto účelu som vytvoril tried *ReConfig*, ktorá je určená na konfiguráciu nástroja bez zásahu do zdrojového kódu. Teraz obsahuje štyri metódy, ktoré sú volané v rôznych častiach nástroja a ktoré načítavajú cesty k externým súborom:

- *getRules()* ku súboru pravidiel
- *getOut_dir()* ku adresáru určenému pre výstup
- *getSchema_output()* ku JSONSchema pre kontrolu výstupu
- *getSchema_rules()* ku JSONSchema pre kontrolu súboru pravidiel

Tieto cesty čítajú z konfiguračného súboru *config.properties* 20, ktorý sa v zdrojových súboroch a v JAR archíve nachádza v *src/main/resources/re/parsers/* adresári. Ich pozmenením je možné čiastočne upraviť funkcionalitu nástroja.

```
#MUST BE either absolute path or a file within a JAR
rules=rules.json
schema_output=schema_output.json
schema_rules=schema_rules.json
out_dir=JSON_output_files
```

Výpis 20: Obsah konfiguračného súboru *config.properties*

Kapitola 7

Testovanie

Táto kapitola popisuje krátke vyhodnotenie výsledného nástroja. V prvej podkapitole 7.1 je popísané testovanie pomocou *JUnit* rámca. Ďalšia podkapitola 7.2 porovnáva rýchlosť nástroja ReFormator so samotne spustenou Tikou pre rovnaké súbory.

7.1 JUnit testy

Ako je spomenuté medzi požiadavkami na nástroj v podkapitole 5.2, je dôležité aby bol výsledný projekt otestovaný na dodaných dokumentoch. K tomu využívam nástroj typický pre testovanie Java aplikácií vo vývoji.

JUnit [3] je rámec pre testovanie jednotiek pre programovací jazyk Java. Je dôležitý pri vývoji riadenom testami a je jedným z rodiny rámcov testovania jednotiek, ktoré sú kolektívne známe ako xUnit. JUnit je v čase kompilácie prepojený ako JAR a je vhodný pre písanie opakovateľných testov.

Nástroj týmito testami prechádza vždy keď je stavaný pomocou *Maven*-u a nie je explicitne zadaný argument *-DskipTests*. Niektoré z dodaných dokumentov, potrebných ku testom, nie je však možné zverejniť. To učiní niektoré testy nepoužiteľnými. Znamená to, že vo výsledne publikovanom nástroji nebudú všetky testy prítomné. Spustiteľné budú iba testy pre klasifikačné triedy "AgilentPDF" a "AgilentXLSX". Platí to ako pre testy validity výstupu tak aj pre klasifikačné testy.

Testovanie rozoznávania dokumentov

Prvá kategória testov bola napísaná pre kontrolu klasifikácie dokumentov. Najprv sú v testoch klasifikované všetky dokumenty, ktoré by mal nástroj poznať. Tieto dokumenty sú rozdelené v adresároch podľa ich klasifikačnej triedy. Pre každý adresár je v cykle porovnávaný výsledok klasifikácie s mojou klasifikáciou pre daný adresár (teda mnou zapísanou hodnotou). Po týchto testoch je otestovaná klasifikácia neznámych dokumentov, tentokrát s očakávaným výsledkom *null*. Okrem klasifikačnej metódy *decideJSON()* je v testoch kontrolovaná aj *decideXML()* metóda, ktorá však po prejdení na JSON metódu nebola udržiavaná a je teda zastaraná ("deprecated"). Pre terajšie dokumenty ešte stále klasifikuje správne.

Testovanie validity výstupných dát

Okrem validácie výstupných dát vždy pri ich vytvorení, som napísal aj testy pre kontrolu validity výstupu pre všetky dodané dokumenty. Testy sú tentokrát rozdelené do Java tried podľa klasifikačnej triedy. Každá Java trieda testuje parser a výstupy jednej klasifikačnej triedy. Výstupy sú porovnávané so správne formátovanými súbormi uloženými v zdrojových súboroch. Súbory sú porovnávané ako polia bytov 21. Taktiež sú výsledky kontrolované oproti *schema_output.json* pre zaručenie správnej štruktúry výstupných dát.

Pre overenie štruktúry, ktorú majú JSON schémy (*schema_output.json* a *schema_rules.json*) vynucovať od výstupných JSON súborov som napísal dva krátke testy. V oboch sú oproti schémam posielané na validáciu chybné JSON súbory a očakáva sa teda ich zlyhanie.

Tieto testy však bohužiaľ nebudú fungovať správne, keďže sú závislé na dodaných dokumentoch. Tieto dokumenty však nie je možné zverejniť na internete. Bez týchto dokumentov nebudú mať testy vstup na spracovanie. Preto bude pre stavanie nástroja potrebné *Maven* spúšťať s prepínačom *-DskipTests*, vďaka ktorému budu pri stavaní testy vynechané.

```
byte[] f1 = Files.readAllBytes(file1.toPath());
byte[] f2 = Files.readAllBytes(file2.toPath());
Assert.assertArrayEquals("Files " + file1.getAbsolutePath() + " and \n" +
↪  file2.getAbsolutePath() + " should be same", f1, f2);
```

Výpis 21: Ukážka porovnávania obsahu dvoch súborov v JUnit testoch

7.2 Testovanie rýchlosti oproti Tika

Keďže sa dá očakávať, že rozšírením Tika funkcionality sa čas spracovania predĺži, tak v tomto teste išlo iba o porovnanie. V prvom teste som porovnal čas spracovania pre všetky dodané dokumenty, ktoré sú známe pre ReFormator. Z výsledkov tohto testu, viditeľných na výpise 22 vyplynulo, že vyvinutý nástroj napojený na Tiku je od jej individuálneho spracovania pomalší o 17,3%.

```
Pre 328 dodaných súborov:
ReFormat.jar = 715.000836852 s
tika-app.jar = 609.259018243 s
Rozdiel      = 105.741818609 s
Tika 1.173-krát rýchlejšia ako ReFormator (o 17,3%).
```

Výpis 22: Výsledky porovnania pri spracovaní všetkých dodaných dokumentov

Druhý test pozostával z 3 častí, kde som v každej porovnal čas spracovania pre rôzne MIME-typy dokumentov pre 1, 50 a 200 dokumentov:

- Pre 1 dokument bol priemerný rozdiel z piatich opakovaní testu menší ako jedna sekunda:

```
Čas spracovania 1 PDF súboru:
        ReFormat.jar = 2.207 s
        tika-app.jar = 1.943 s
        Rozdiel      = 0.264 s
        Tika 1.135-krát rýchlejšia ako ReFormator (o 13,5%).
Čas spracovania 1 CSV súboru:
        ReFormat.jar = 1.765 s
        tika-app.jar = 1.280 s
        Rozdiel      = 0.485 s
        Tika 1.378-krát rýchlejšia ako ReFormator (o 37,8%).
Čas spracovania 1 XLSX súboru:
        ReFormat.jar = 2.771 s
        tika-app.jar = 2.182 s
        Rozdiel      = 0.589 s
        Tika 1.269-krát rýchlejšia ako ReFormator (o 26,9%).
```

Výpis 23: Výsledky porovnania pri spracovaní jedného dokumentu

- Pre 50 dokumentov sa už priemerný rozdiel pre 3 opakovania pohyboval v desiatkach sekúnd:

```
Čas spracovania 50 PDF súborov:
        ReFormat.jar = 110.775 s
        tika-app.jar = 97.314 s
        Rozdiel      = 13.461 s
        Tika 1.138-krát rýchlejšia ako ReFormator (o 13,8%).
Čas spracovania 50 CSV súborov:
        ReFormat.jar = 88.276 s
        tika-app.jar = 68.464 s
        Rozdiel      = 19.812 s
        Tika 1.289-krát rýchlejšia ako ReFormator (o 28,9%).
Čas spracovania 50 XLSX súborov:
        ReFormat.jar = 146.038 s
        tika-app.jar = 114.351 s
        Rozdiel      = 31.687 s
        Tika 1.277-krát rýchlejšia ako ReFormator (o 27,7%).
```

Výpis 24: Výsledky porovnania pri spracovaní päťdesiatich dokumentov

- Pre 200 dokumentov sa už rozdiel pre jeden test pohyboval v desiatkach sekúnd:

```
Čas spracovania 200 PDF súborov:
        ReFormat.jar = 413.674 s
        java-app.jar = 368.565 s
        Rozdiel      = 45.109 s
        Tika 1.122-krát rýchlejšia ako ReFormator (o 12,2%).
Čas spracovania 200 CSV súborov:
        ReFormat.jar = 332.816 s
        java-app.jar = 259.387 s
        Rozdiel      = 73.429 s
        Tika 1.283-krát rýchlejšia ako ReFormator (o 28,3%).
Čas spracovania 200 XLSX súborov:
        ReFormat.jar = 575.233 s
        java-app.jar = 445.024 s
        Rozdiel      = 130.209 s
        Tika 1.292-krát rýchlejšia ako ReFormator (o 29,2%).
```

Výpis 25: Výsledky porovnania pri spracovaní dvesto dokumentov

Z týchto výsledkov vyplynulo, že časový rozdiel v spracovaní pre malý počet dokumentov 23 je skoro nepoznateľný a ide o desatiny sekundy pre všetky média typy. Pri 50 dokumentoch 24 sa už rozdiel pohybuje v desiatkach sekúnd. Taktiež je možné vidieť rozdiel medzi PDF súbormi oproti CSV a XLSX súborom (najskôr z dôvodu rozdielneho prístupu k spracovaniu dokumentov pre XLSX 6.4 a počtu riadkov v CSV súboroch 6.4). Pri 200 súboroch 25 je tento rozdiel už jasne viditeľný (12,2% spomalenie pre PDF oproti 29,3% spomaleniu pre XLSX). Časový rozdiel sa tu už dostáva k hodnotám okolo sto sekúnd.

Vyhodnotenie

Pre dodané dokumenty spĺňa nástroj, až na malé detaily, svoju mienenú funkcionalitu a prechádza aj všetkými testami, ktoré som napísal. Niektoré malé nedokonalosti výsledného projektu, ktoré som si všimol, sú napríklad: občasné zlé identifikovanie hodnoty kľúča, pre niektoré kľúče skrátenie hodnoty ak ide cez viacero riadkov, chybná extrakcia dát v prípade prítomnosti hodnoty kľúča ne nečakanom mieste (napr. medzi hodnotami v tabuľke).

Vypočítaný priemerný percentuálny rozdiel rýchlosti bol nakoniec 23,56%, čo je podľa mňa uspokojivý výsledok, keď uvážim, že nástroj naviac pre známe dokumenty robí klasifikáciu, extrakciu, štrukturalizáciu a validáciu dát. A to všetko navyše ku výstupu od Tiky, ktorý je možné tiež do určitej miery ovplyvniť dopísaním argumentov ku príkazu do spúšťacieho skriptu. Tieto argumenty budú spracované ako by ich spracovala Tika keďže ako hlavná trieda je nastavená *org.apache.tika.cli.TikaCLI*.

Kapitola 8

Záver

Cieľom bakalárskej práce bolo navrhnúť a implementovať nástroj, ktorý by rozširoval funkcionalitu Apache Tika nástroja o extrakciu dôležitých dát a ich štrukturalizáciu na výstupe. Konečný produkt — ReFormator — ponúka klasifikáciu dokumentov na základe ich obsahu a metadát, rozšíriteľnosť o nové triedy dokumentov bez zásahu do zdrojových kódov, štrukturalizáciu dát pre triedy známe dokumentu a validáciu štruktúry výstupných dát.

Nástroj bol implementovaný ako JAR aplikácia, ktorú je možné samostatne spustiť alebo pridať ako závislosť do iného projektu. Okrem toho ponúka konfiguráciu pomocou *Java properties file*. To všetko umožňuje nástroj flexibilne používať s vlastnou implementáciou a v budúcnosti veľmi jednoducho rozšíriť.

Nástroj bol úspešne zapojený do pracovného postupu Tiky ,z ktorej využíva detekciu MIME typov a parsovanie na extrakciu textových dát a metadát pre jeho klasifikačný proces.

ReFormator je schopný rozoznávať dokumenty, ktoré sú popísané v súbore pravidiel a je pre nich vytvorený parser, ktorému ich potom predajú na parseru na extrakciu textu. Aj keď nástroj dokument nespozná, na výstup vždy vypíše obsah a metadáta celého dokumentu získané Tikou. Výstup je reťazec v XHTML formáte.

Konečný produkt bol podrobený jednotkovým testom pre overenie jeho funkčnosti a výstupov. Tieto testy využili všetky dodané dokumenty k vývoju, pre ktoré bol nástroj implementovaný. Taktiež bol podrobený niekoľkým testom pre rýchlosť implementácie, aby bol čo najvýkonnejší a čo mal čo najmenšie spomalenie oproti Tike.

Nástroj by do budúcna mohol byť rozšírený o podporu ďalších klasifikačných tried a parserov im príslušných. Bol navrhnutý a implementovaný s pomyslením na túto skutočnosť a preto je na to prispôsobený.

Literatúra

[1] Ambler, S. W. *Relational databases 101: Looking at the whole picture* [online]. [cit. 2021-05-04]. Dostupné z: http://www.agiledata.org/essays/relationalDatabases.html.

[2] Aristarán, M. a Tigas, M. Introducing Tabula. [online]. 2013, [cit. 2021-05-04]. Dostupné z: https://source.opennews.org/articles/introducing-tabula/.

[3] Beck, K. *JUnit Pocket Guide*. Sebastopol, CA: O'Reilly Media, 2004 [cit. 2021-05-04]. ISBN 9780596007430. Dostupné z: https://en.wikipedia.org/w/index.php?title=JUnit&oldid=1016141285.

[4] Borenstein, N. S. a Freed, N. *Multipurpose internet mail extensions (MIME) part one: Format of internet message bodies* [online]. 1996 [cit. 2021-05-04]. Dostupné z: https://tools.ietf.org/html/rfc2045.

[5] Borenstein, N. S. a Freed, N. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types* [online]. 1996 [cit. 2021-05-04]. Dostupné z: https://tools.ietf.org/html/rfc2046.

[6] Bray, T. *The JavaScript object notation (JSON) data interchange format* [online]. [cit. 2021-05-04]. Dostupné z: https://tools.ietf.org/html/rfc7159.

[7] Chet, R. a Brian, F. *Bash Reference Manual: Edition 2.2*. 2.2. IUniverse, 2000. ISBN 9780595100309. Dostupné z: https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html.

[8] Gavin, B. What is an XLSX file (and how do I open one)? [online]. How-To Geek. Október 2018, [cit. 2021-05-05]. Dostupné z: https://www.howtogeek.com/392333/what-is-an-xlsx-file-and-how-do-i-open-one/.

[9] Gregersen, E. Data structure. [online]. Apríl 2017, [cit. 2021-05-04]. Dostupné z: https://www.britannica.com/technology/data-structure.

[10] Grune, D. a Jacobs, C. J. H. *Parsing techniques: A practical guide*. 2. vyd. New York, NY: Springer, 2010 [cit. 2021-05-04]. ISBN 9781441919014. Dostupné z: https://www.springer.com/gp/book/9780387202488#otherversion=9781441919014.

[11] *Lesson: Packaging programs in JAR files* [online]. [cit. 2021-05-04]. Dostupné z: https://docs.oracle.com/javase/tutorial/deployment/jar/.

[12] *JAR File Specification* [online]. [cit. 2021-05-04]. Dostupné z: https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jar.html.

[13] Mattmann, C. a Zitting, J. *Tika in Action*. USA: Manning Publications Co., 2011. ISBN 978-1-935182-85-6. Dostupné z: https://www.manning.com/books/tika-in-action.

[14] *Npm: pdf2json* [online]. [cit. 2021-05-06]. Dostupné z: https://www.npmjs.com/package/pdf2json.

[15] Porter, B., Zyl, J. van a Lamy, O. *Maven – welcome to Apache maven* [online]. [cit. 2021-05-04]. Dostupné z: https://maven.apache.org/.

[16] *How to reuse inputStream?* [online]. 2019 [cit. 2021-05-04]. Dostupné z: https://developpaper.com/how-to-reuse-inputstream/.

[17] *ServiceLoader (Java Platform SE 6)* [online]. 2015 [cit. 2021-05-04]. Dostupné z: https://docs.oracle.com/javase/6/docs/api/java/util/ServiceLoader.html.

[18] Seth, R., Dahal, B., Sitikhu, P. a Adhikari, S. *Intelligent Data Capture and Extraction with OCR and AI - Docsumo* [online]. [cit. 2021-05-04]. Dostupné z: https://docsumo.com/.

[19] Shafranovich, Y. *Common Format and MIME Type for Comma-Separated Values (CSV) Files* [online]. 2005 [cit. 2021-05-04]. Dostupné z: https://tools.ietf.org/html/rfc4180.

[20] Shreeshrii, Weil, S. a Pösel, R. *Tessdoc* [online]. 2021 [cit. 2021-05-04]. Dostupné z: https://github.com/tesseract-ocr/tessdoc.

[21] *Introduction to the service provider interfaces* [online]. [cit. 2021-05-04]. Dostupné z: https://docs.oracle.com/javase/tutorial/sound/SPI-intro.html.

[22] *Validované datové úložiště - CEP - TA ČR Starfos* [online]. [cit. 2021-05-03]. Dostupné z: https://starfos.tacr.cz/cs/project/FV40176.

[23] Tomassetti, G. *A guide to parsing: Algorithms and terminology* [online]. [cit. 2021-05-04]. Dostupné z: https://tomassetti.me/guide-parsing-algorithms-terminology/.

[24] *Weka Tutorial - Tutorialspoint* [online]. [cit. 2021-05-04]. Dostupné z: https://www.tutorialspoint.com/weka/index.htm.

[25] Westerlund, M., Cheshire, S., Touch, J., Eggert, L. a Cotton, M. *Internet assigned numbers authority (IANA) procedures for the management of the service name and transport protocol port number registry*. 2011 [cit. 2021-05-05]. Dostupné z: https://tools.ietf.org/html/rfc6335.

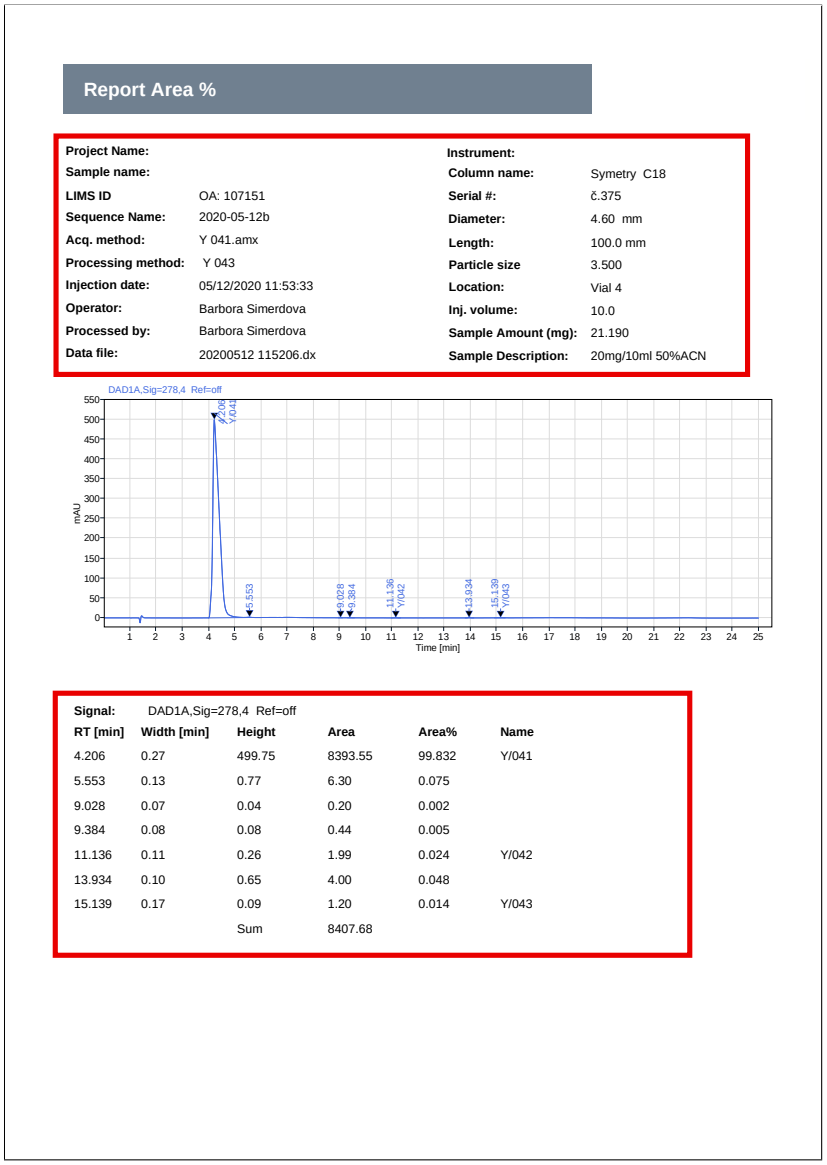
[26] Wright, A., Andrews, H., Hutton, B. a Dennis, G. *Draft-bhutton-json-schema-00 - JSON schema: A Media Type for Describing JSON Documents* [online]. 2020 [cit. 2021-05-04]. Dostupné z: https://datatracker.ietf.org/doc/draft-bhutton-json-schema/.

[27] Zilles, S., Masinter, L., Pravetz, J. D. a Taft, E. A. *The application/pdf Media Type*. 2004 [cit. 2021-05-04]. Dostupné z: https://tools.ietf.org/html/rfc3778.

[28] *ZIP* [online]. [cit. 2021-05-06]. Dostupné z:
https://pkware.cachefly.net/webdocs/APPNOTE/APPNOTE-6.3.9.TXT.

Príloha A

Ukážka dokumentu z dodaných dát



Report Area %

Project Name:		**Instrument:**	
Sample name:		**Column name:**	Symetry C18
LIMS ID	OA: 107151	**Serial #:**	č.375
Sequence Name:	2020-05-12b	**Diameter:**	4.60 mm
Acq. method:	Y 041.amx	**Length:**	100.0 mm
Processing method:	Y 043	**Particle size**	3.500
Injection date:	05/12/2020 11:53:33	**Location:**	Vial 4
Operator:	Barbora Simerdova	**Inj. volume:**	10.0
Processed by:	Barbora Simerdova	**Sample Amount (mg):**	21.190
Data file:	20200512 115206.dx	**Sample Description:**	20mg/10ml 50%ACN

Signal: DAD1A,Sig=278,4 Ref=off

RT [min]	Width [min]	Height	Area	Area%	Name
4.206	0.27	499.75	8393.55	99.832	Y/041
5.553	0.13	0.77	6.30	0.075	
9.028	0.07	0.04	0.20	0.002	
9.384	0.08	0.08	0.44	0.005	
11.136	0.11	0.26	1.99	0.024	Y/042
13.934	0.10	0.65	4.00	0.048	
15.139	0.17	0.09	1.20	0.014	Y/043
		Sum	8407.68		

Obr. A.1: Screenshot jedného z dodaných dokumentov klasifikačnej triedy Agilent (niektoré údaje boli zakryté na želanie spolupracujúcej firmy)

Title:	QC Area% basic
Location:	file:///home/rene/Downloads/showcase.pdf
Subject:	Single Injection Report
Author:	Barbora Simerdova
Keywords:	*None*
Producer:	iTextSharp™ 5.5.7 ©2000-2015 iText Group NV (Agilent Technologies, Inc; licensed version)
Creator:	Agilent Technologies
Created:	Ut 12. máj 2020, 12:24:49
Modified:	Ut 12. máj 2020, 12:24:49
Format:	PDF-1.5
Number of Pages:	1
Optimized:	No
Security:	No

Obr. A.2: Niektoré metadáta dokumentu z obrázku A

Príloha B

Ukážka schémy výstupných dát

```
{
  "$schema": "http://json-schema.org/schema#",
  "title": "OutputSechema",
  "description": "Schema for validating this project's outputs",
  "type": "object",
  "properties": {
    "content": {
      "type": "array",
      "items": {
        "$ref": "#/$defs/part"
      }
    },
    "metadata": {
      "type": "object",
      "patternProperties": {
        ".*": {
          "type": ["string", "number", "boolean", "null"]
        }
      },
      "additionalProperties": false
    }
  },
  "required": ["content", "metadata"],
  "additionalProperties": false,
}
```

Výpis 26: Prvá časť schémy. Popisuje objekty na najvyššej úrovni.

```
{
  "$defs": {
    "part": {
      "type": "object",
      "properties":{
        "section_name": {
          "type": "string"
        },
        "data": {
          "type": "object",
          "patternProperties": {
            ".*": {
              "type": ["string", "number", "boolean", "null"]
            }
          }
        }
      },
      "required": ["data"],
      "patternProperties": {
        "^table(_[1-9]+[0-9]*)?$": {
          "type": "object"
        }
      },
      "minProperties": 1,
      "additionalProperties": false
    }
  }
}
```

Výpis 27: Druhá časť schémy. Popisuje objekty vnorené v *content* poli.

Príloha C

Ukážka výstupu nástroja


```
{
  "content": [
    {
      "section_name": "Report 1",
      "data": {
        "Project Name": "",
        "Instrument": "",
        "Sample name": "",
        "Column name": "Symetry  C18",
        "LIMS ID OA": "107151",
        "Serial #": "č.375",
        "Sequence Name": "2020-05-12b",
        "Diameter": "4.60  mm",
        "Acq. method": "Y 041.amx",
        "Length": "100.0 mm",
        "Processing method": "Y 043",
        "Particle size": "3.500",
        "Injection date": "05/12/2020 11:53:33",
        "Location": "Vial 4",
        "Operator": "Barbora Simerdova",
        "Inj. volume": "10.0",
        "Processed by": "Barbora Simerdova",
        "Sample Amount (mg)": "21.190",
        "Data file": "20200512 115206.dx",
        "Sample Description": "20mg/10ml 50%ACN"
      },
      "table_1": {
        "Sum": {
          "Area": "8407.68"
        },
        "Signal": "DAD1A,Sig=278,4  Ref=off",
        "1": {
          "RT [min]": "4.206",
          "Width [min]": "0.27",
          "Height": "499.75",
```

```
    "Area": "8393.55",
    "Area%": "99.832",
    "Name": "Y/041"
  },
  "2": {
    "RT [min]": "5.553",
    "Width [min]": "0.13",
    "Height": "0.77",
    "Area": "6.30",
    "Area%": "0.075",
    "Name": ""
  },
  "3": {
    "RT [min]": "9.028",
    "Width [min]": "0.07",
    "Height": "0.04",
    "Area": "0.20",
    "Area%": "0.002",
    "Name": ""
  },
  "4": {
    "RT [min]": "9.384",
    "Width [min]": "0.08",
    "Height": "0.08",
    "Area": "0.44",
    "Area%": "0.005",
    "Name": ""
  },
  "5": {
    "RT [min]": "11.136",
    "Width [min]": "0.11",
    "Height": "0.26",
    "Area": "1.99",
    "Area%": "0.024",
    "Name": "Y/042"
  },
  "6": {
    "RT [min]": "13.934",
    "Width [min]": "0.10",
    "Height": "0.65",
    "Area": "4.00",
    "Area%": "0.048",
    "Name": ""
  },
  "7": {
    "RT [min]": "15.139",
    "Width [min]": "0.17",
    "Height": "0.09",
```

```
        "Area": "1.20",
        "Area%": "0.014",
        "Name": "Y/043"
      }
    }
  }
],
"metadata": {
  "pdf:unmappedUnicodeCharsPerPage": "0",
  "pdf:PDFVersion": "1.5",
  "dc:description": "Single Injection Report",
  "xmp:CreatorTool": "Agilent Technologies",
  "pdf:docinfo:title": "QC Area% basic",
  "pdf:hasXFA": "false",
  "access_permission:modify_annotations": "true",
  "access_permission:can_print_degraded": "true",
  "dc:creator": "Barbora Simerdova",
  "dcterms:created": "2020-05-12T10:24:49Z",
  "dcterms:modified": "2020-05-12T10:24:49Z",
  "dc:format": "application/pdf; version=1.5",
  "pdf:docinfo:creator_tool": "Agilent Technologies",
  "access_permission:fill_in_form": "true",
  "pdf:docinfo:modified": "2020-05-12T10:24:49Z",
  "pdf:encrypted": "false",
  "dc:title": "QC Area% basic",
  "xmp:CreateDate": "2020-05-12T12:24:49Z",
  "cp:subject": "Single Injection Report",
  "Content-Length": "163479",
  "pdf:docinfo:subject": "Single Injection Report",
  "pdf:hasMarkedContent": "false",
  "Content-Type": "application/pdf",
  "xmp:ModifyDate": "2020-05-12T12:24:49Z",
  "pdf:docinfo:creator": "Barbora Simerdova",
  "pdf:producer": "iTextSharp 5.5.7 ©2000-2015 iText Group NV (Agilent
  ↪  Technologies, Inc; licensed version)",
  "dc:subject": "Single Injection Report",
  "access_permission:extract_for_accessibility": "true",
  "access_permission:assemble_document": "true",
  "xmpTPg:NPages": "1",
  "resourceName": "107151-56-2020-05-12 107151.pdf",
  "pdf:hasXMP": "true",
  "pdf:charsPerPage": "966",
  "access_permission:extract_content": "true",
  "access_permission:can_print": "true",
  "X-TIKA:Parsed-By": "org.apache.tika.parser.CompositeParser",
  "access_permission:can_modify": "true",
  "pdf:docinfo:producer": "iTextSharp 5.5.7 ©2000-2015 iText Group NV
  ↪  (Agilent Technologies, Inc; licensed version)",
```

```
    "pdf:docinfo:created": "2020-05-12T10:24:49Z"
  }
}
```

Príloha D

README

ReFormator - Data and Table extraction tool

Tika extension for extracting data created in JAVA

Table of contents

General info

This project is an easy to extend tool for extracting data from documents and their structuring into a JSON file. It uses Tika to detect MIME types and own classifier for further document classification into classes.

Technologies

Project is created with: * Java: 15 * Maven: 3.6.3 * JDK: 8 and higher

Setup

To build and run this project from scratch you need to have all the necessary dependencies downloaded. You may need to alter versions of the dependencies in the pom.xml if you have newer/older versions, or download some.

If you took care of the dependencies than extract the contents from zip file and follow the commands to build the project:

```
To skip tests add -DskipTests after mvn command
$ unzip ReFormat.zip
$ cd ReFormat/Modules/
$ sudo apt update
$ sudo apt install maven
$ mvn clean compile assembly:single
```

For running the program JDK is necessary to load classess within JAR archive. Than to run the program using written scripts:

```
$ cd ..
$ chmod 777 parse.sh                    # if on Linux
$ ./parse.sh/.bat <file_to_parse>       # sh/bat depending on your OS
```

To run downloaded jar alone you need to provide it with TikaConfig file from the project, because it needs to be given to TikaCLI class as argument:

```
$ cd <directory_containing_ReFormat.jar>
$ java -jar ReFormat.jar --config=<path_to_TikaConfigFile>/<TikaConfigFile> <file_to_parse>
```

If calling from ReFormat directory of the project:

```
$ java -jar ReFormat.jar --config=Modules/src/main/resources/re/parsers/tika-config-bc.xml <file_to_parse>
```

To configurate ReFormator behaviour alter config.properties file within the JAR archive or in sources before building the project:

```
rules=rules.json                  # to change file with rules
schema_output=schema_output.json  # to change schema controlling structure of the output
schema_rules=schema_rules.json    # to change schema controlling structure of the rules file
out_dir=JSON_output_files         # to change output directory
```